Ansible Tower Administration Guide

Release Ansible Tower 3.0.3

Red Hat, Inc.

Jun 06, 2017

CONTENTS

1 Tower Licensing, Updates, and Support **2**
- 1.1 Support . . . 2
- 1.2 Trial / Evaluation . . . 3
- 1.3 Subscription Types . . . 3
- 1.4 Node Counting in Licenses . . . 3
- 1.5 License Features . . . 4
- 1.6 Tower Component Licenses . . . 4

2 Starting, Stopping, and Restarting Tower **5**

3 Custom Inventory Scripts **6**
- 3.1 Writing Inventory Scripts . . . 8

4 Management Jobs **9**
- 4.1 Removing Old Activity Stream Data . . . 9
- 4.2 Removing Old Fact (System Tracking) Data . . . 13
- 4.3 Removing Old Job History . . . 16

5 Redundancy with Tower **21**
- 5.1 Setup Considerations . . . 22
- 5.2 Differences between Primary and Secondary Instances . . . 24
- 5.3 Post-Installation Changes to Primary Instances . . . 24
- 5.4 Examining the redundancy configuration of Tower . . . 24
- 5.5 Promoting a Secondary Instance/Failover . . . 25
- 5.6 Decommissioning Secondary instances . . . 26

6 Proxy Support **27**
- 6.1 Reverse Proxy . . . 27

7 Tower Logfiles **28**

8 The *tower-manage* Utility **29**
- 8.1 Inventory Import . . . 29
- 8.2 Cleanup of old data . . . 29
- 8.3 HA management . . . 30

9 LDAP Integration **31**
- 9.1 LDAPS . . . 33
- 9.2 Debugging . . . 33
- 9.3 Referrals . . . 33

10 Setting up Authentication **35**
10.1 Basic Authentication Settings: . . . 35
10.2 Google OAuth2 Settings . . . 35
10.3 Github OAuth2 Settings . . . 36
10.4 SAML Authentication Settings . . . 37
10.5 RADIUS Authentication Settings . . . 38
10.6 Using LDAP with Tower . . . 39
10.7 Organization and Team Mapping . . . 39

11 Changing the Default Timeout for Authentication **42**

12 User Authentication with Kerberos **44**
12.1 AD and Kerberos Credentials . . . 45
12.2 Working with Kerberos Tickets . . . 46

13 Working with Session Limits **47**

14 Backing Up and Restoring Tower **48**
14.1 Backup/Restore Playbooks . . . 48
14.2 Backup and Restoration Considerations . . . 49

15 Using Custom Logos in Ansible Tower **50**

16 Troubleshooting Tower **52**
16.1 Error logs . . . 52
16.2 Problems connecting to your host . . . 52
16.3 Changing the WebSockets port for live events . . . 52
16.4 Problems running a playbook . . . 53
16.5 Problems when running a job . . . 53
16.6 Playbooks aren't showing up in the "Job Template" drop-down . . . 53
16.7 Playbook stays in pending . . . 53
16.8 Cancel a Tower job . . . 53
16.9 Reusing an external HA database causes installations to fail . . . 54
16.10 PRoot functionality and variables . . . 54
16.11 Private EC2 VPC Instances in Tower Inventory . . . 54
16.12 Troubleshooting "Error: provided hosts list is empty" . . . 55

17 Tower Tips and Tricks **56**
17.1 Using the Tower CLI Tool . . . 56
17.2 Launching a Job Template via the API . . . 56
17.3 `tower-cli` Job Template Launching . . . 58
17.4 Changing the Tower Admin Password . . . 58
17.5 Creating a Tower Admin from the commandline . . . 58
17.6 Setting up a jump host to use with Tower . . . 59
17.7 View Ansible outputs for JSON commands when using Tower . . . 59
17.8 Locate and configure the Ansible configuration file . . . 59
17.9 View a listing of all ansible_ variables . . . 59
17.10 Using virtualenv with Ansible Tower . . . 60
17.11 Configuring the `towerhost` hostname for notifications . . . 60
17.12 Launching Jobs with curl . . . 60
17.13 Dynamic Inventory and private IP addresses . . . 61
17.14 Filtering instances returned by the dynamic inventory sources in Tower . . . 61
17.15 Using an unreleased module from Ansible source with Tower . . . 62
17.16 Using callback plugins with Tower . . . 62
17.17 Connecting to Windows with winrm . . . 62

17.18 Importing existing inventory files and host/group vars into Tower 63

18 Introduction to tower-cli **64**
18.1 License 64
18.2 Capabilities 64
18.3 Installation 64
18.4 Configuration 65

19 Usability Analytics and Data Collection **68**

20 Postface **69**

21 Index **72**

22 Copyright © 2016 Red Hat, Inc. **73**

Index **74**

Thank you for your interest in Ansible Tower by Red Hat. 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 3.0.3.

Ansible Tower Version 3.0.3; October 28, 2016; 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.

- **Self-Support**
  - Manage smaller environments (up to 250 Managed Nodes)
  - Maintenance and upgrades included
  - No support or SLA included
- **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:

- 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

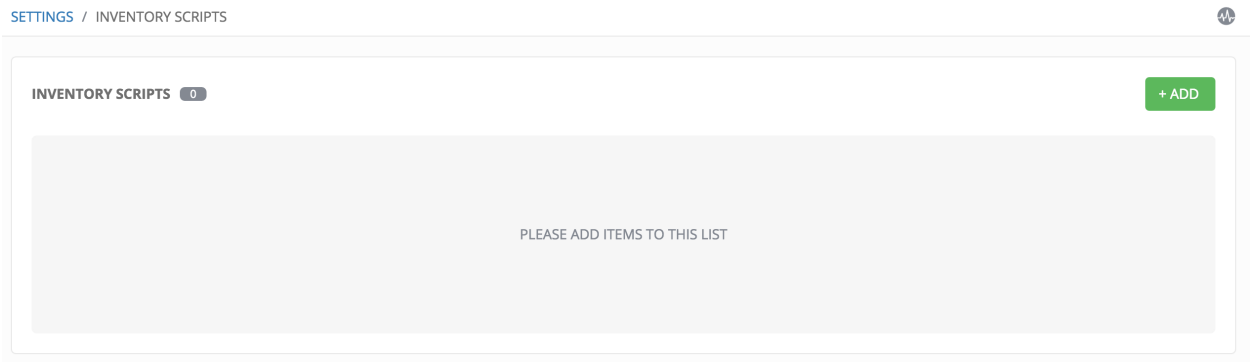
THREE

CUSTOM INVENTORY SCRIPTS

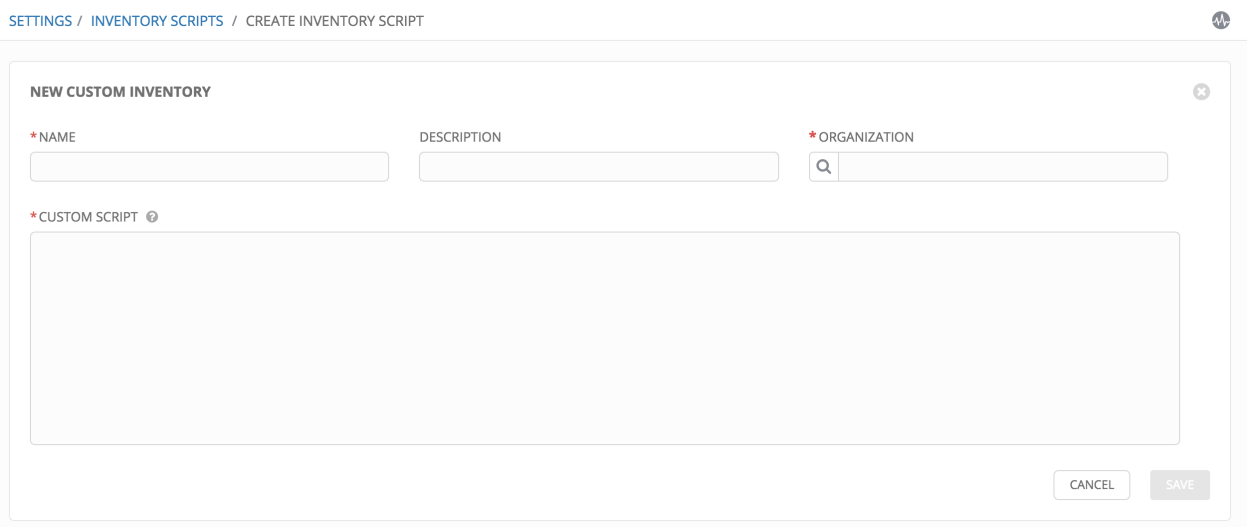
Tower includes built-in support for syncing dynamic inventory from cloud sources such as Amazon AWS, Google Compute Engine, and Rackspace, among others. Tower also offers the ability to use a custom script to pull from your own inventory source.

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.

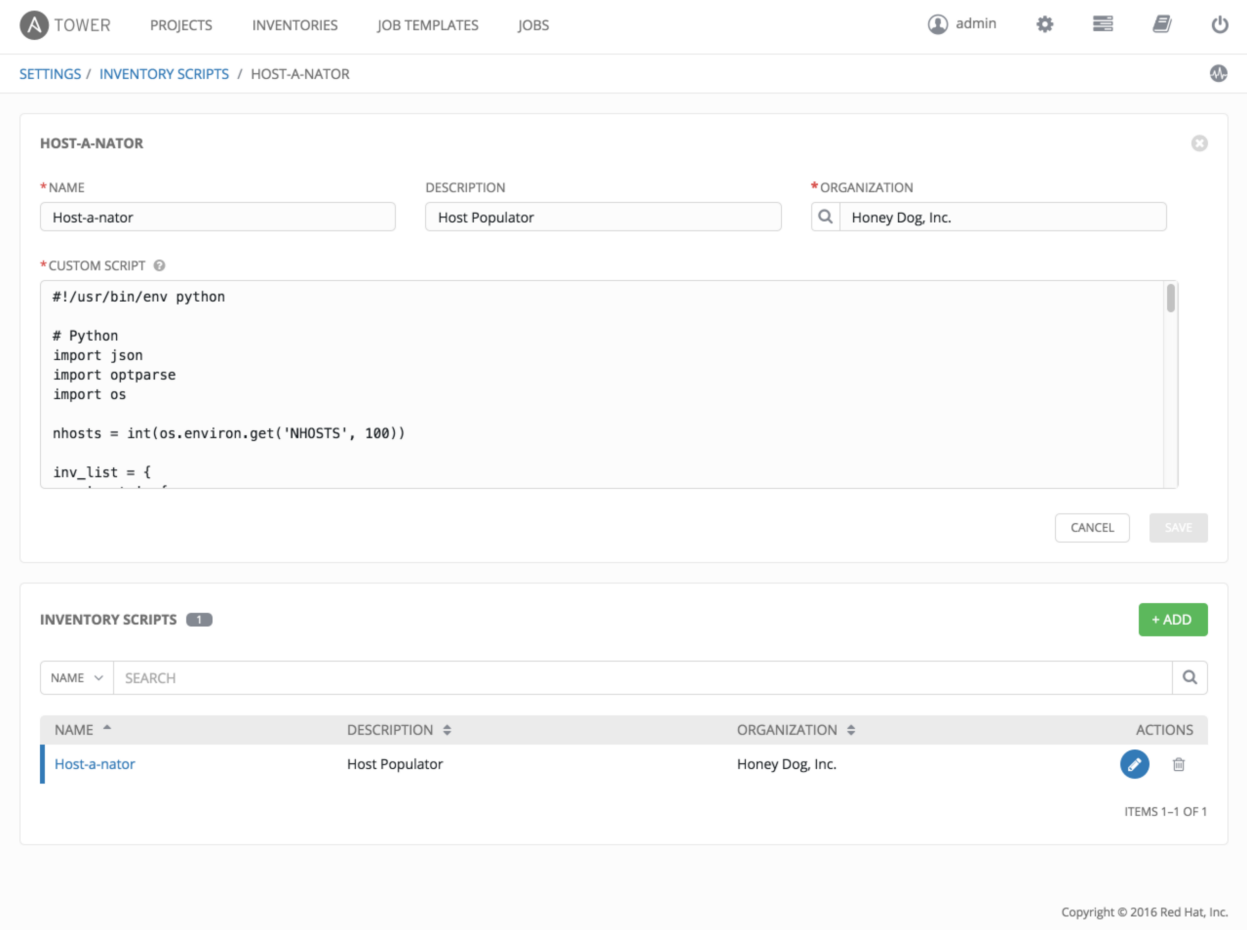


To add a new custom inventory script, click the + ADD button.



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.



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`.

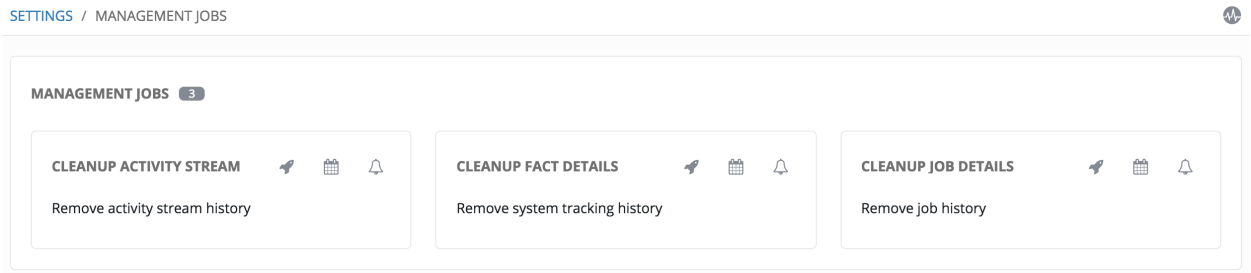
For more information on dynamic inventory scripts and how to write them, refer to the Intro to Dynamic Inventory and Developing Dynamic Inventory Sources sections of the Ansible documentation, or review the example dynamic inventory scripts on GitHub.

CHAPTER

FOUR

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**.

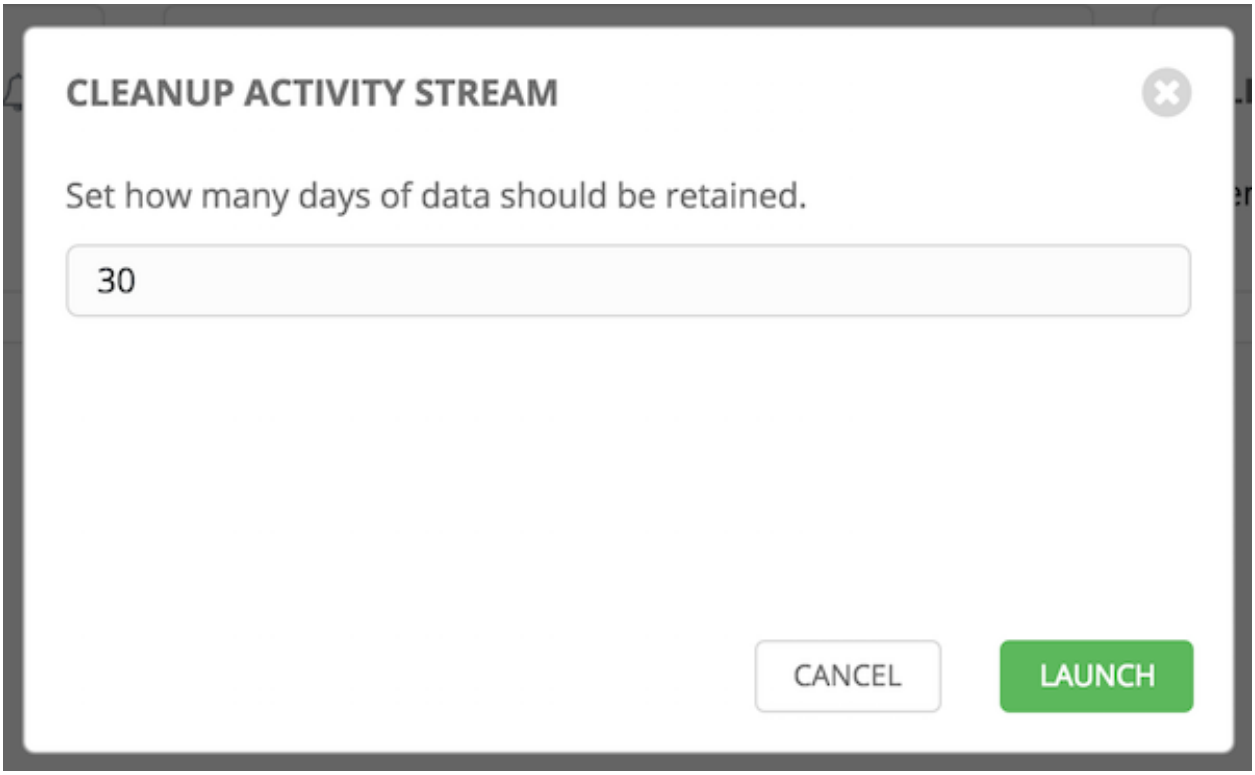


Several job types are available for you to schedule and launch:

- **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

4.1 Removing Old Activity Stream Data

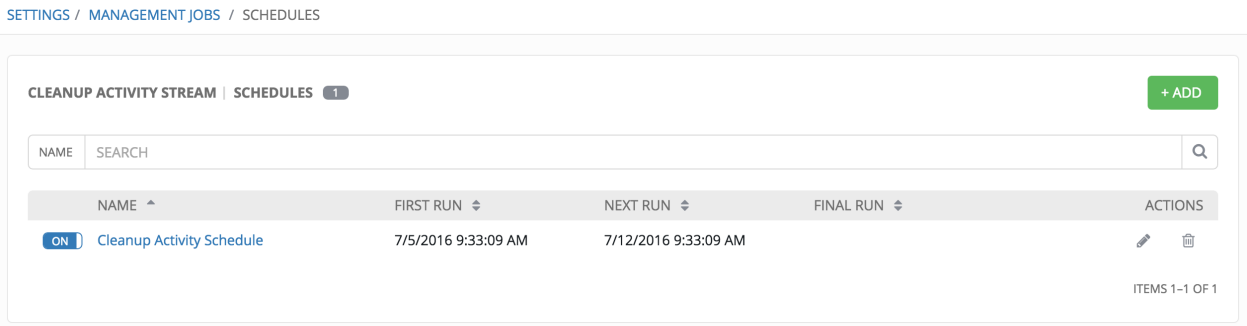
To remove older activity stream data, click on the button beside **Cleanup Activity Stream**.



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

4.1.1 Scheduling

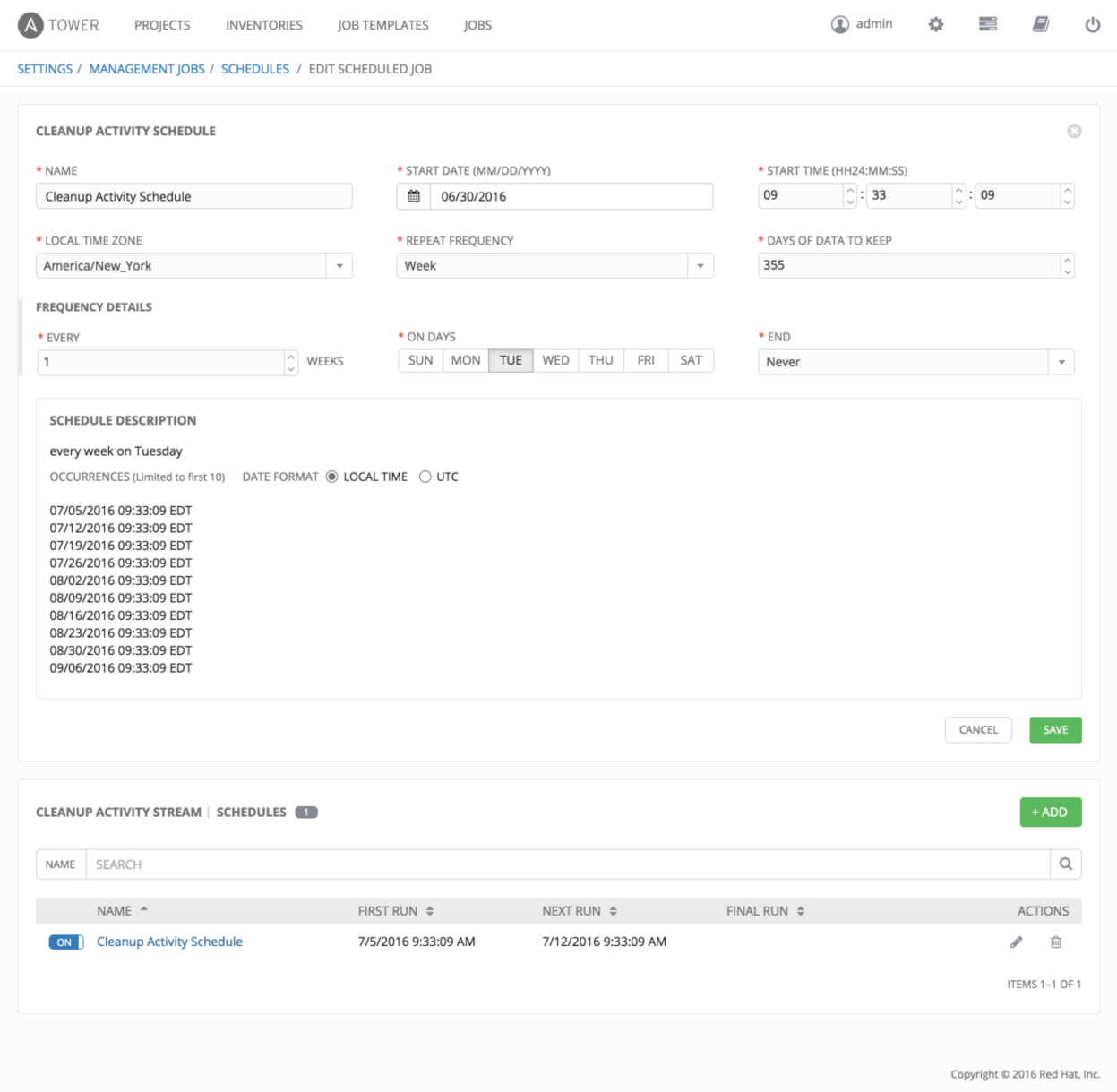
To review or set a schedule for purging data marked for deletion, click on the button.



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

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

button to create a new schedule for this management job.



Enter the appropriate details into the following fields and 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.

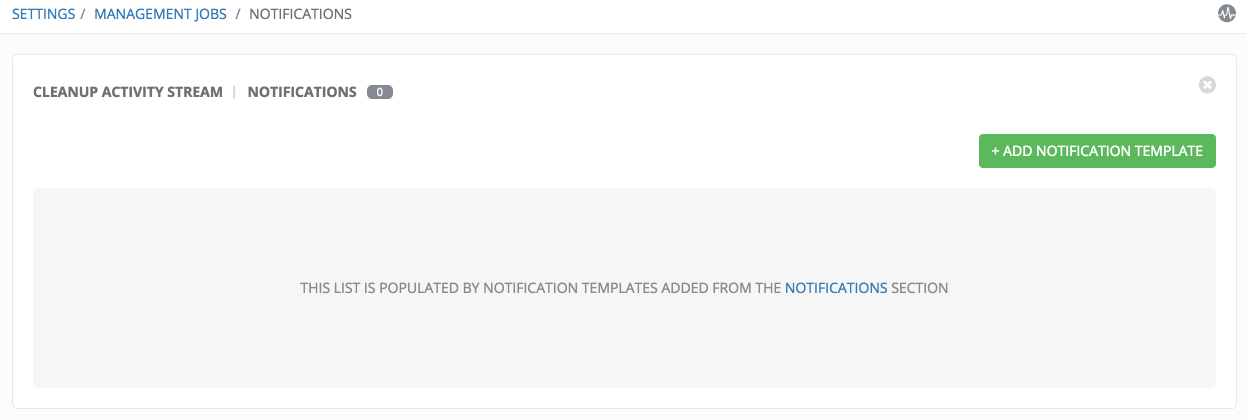
Note: Jobs are scheduled in UTC. Repeating jobs that runs at a specific time of day may move relative to a local

timezone when Daylight Saving Time shifts occur.

4.1.2 Notifications

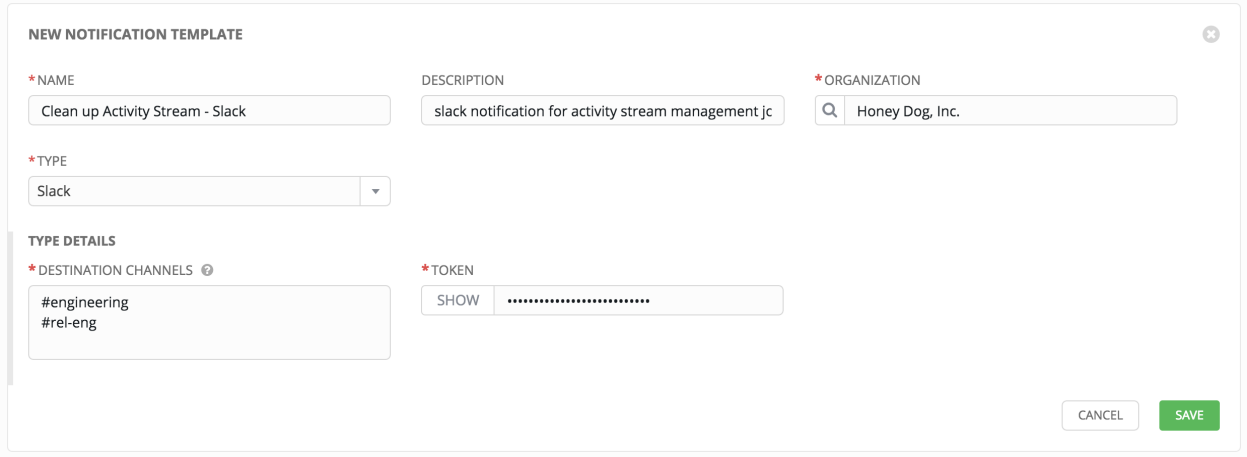
To set or review notifications associated with this management job, click the Configure Notifications ( ) button.

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



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

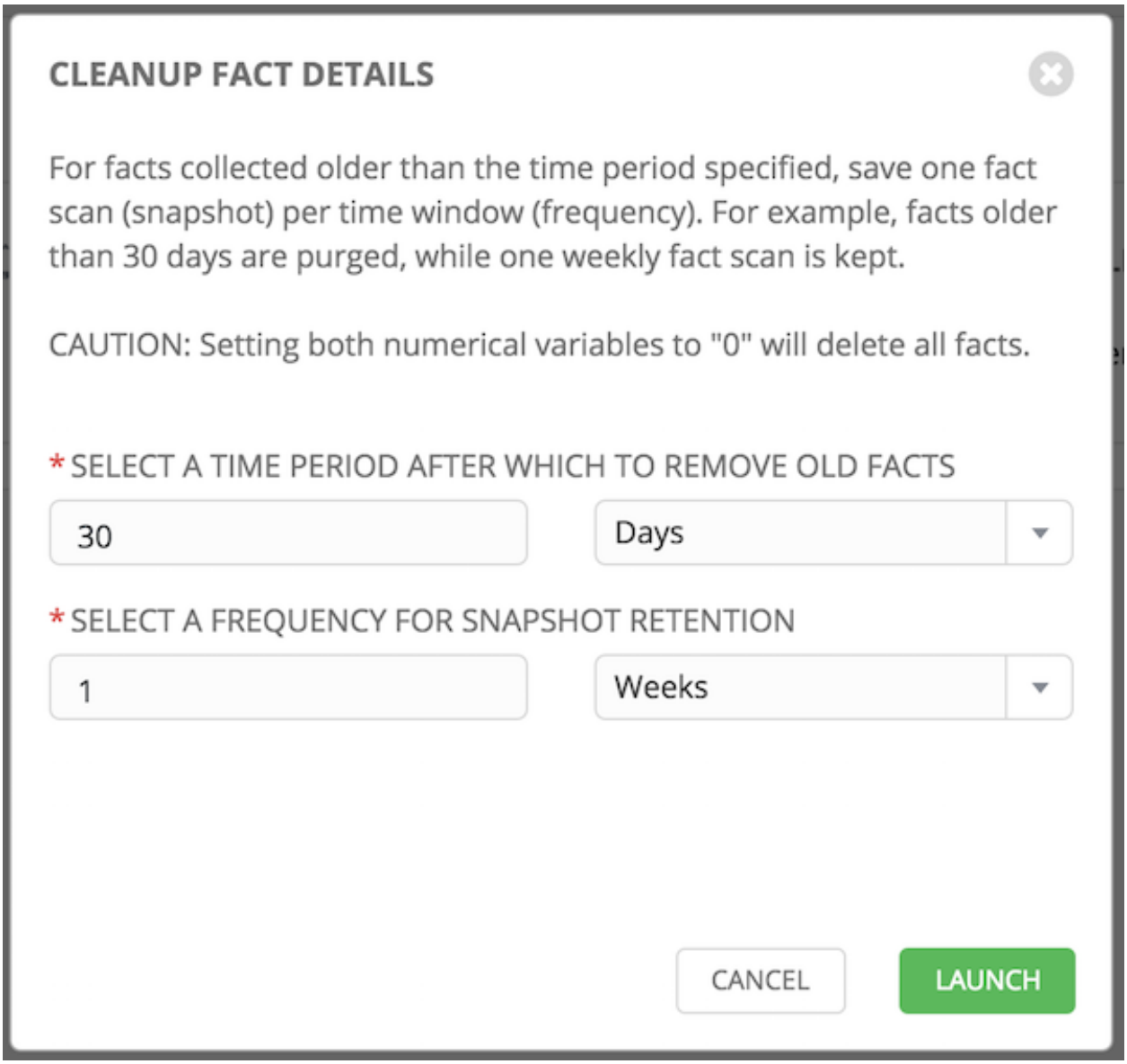
- Email
- Slack
- Twilio
- PagerDuty
- HipChat
- Webhook
- IRC



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

4.2 Removing Old Fact (System Tracking) Data

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

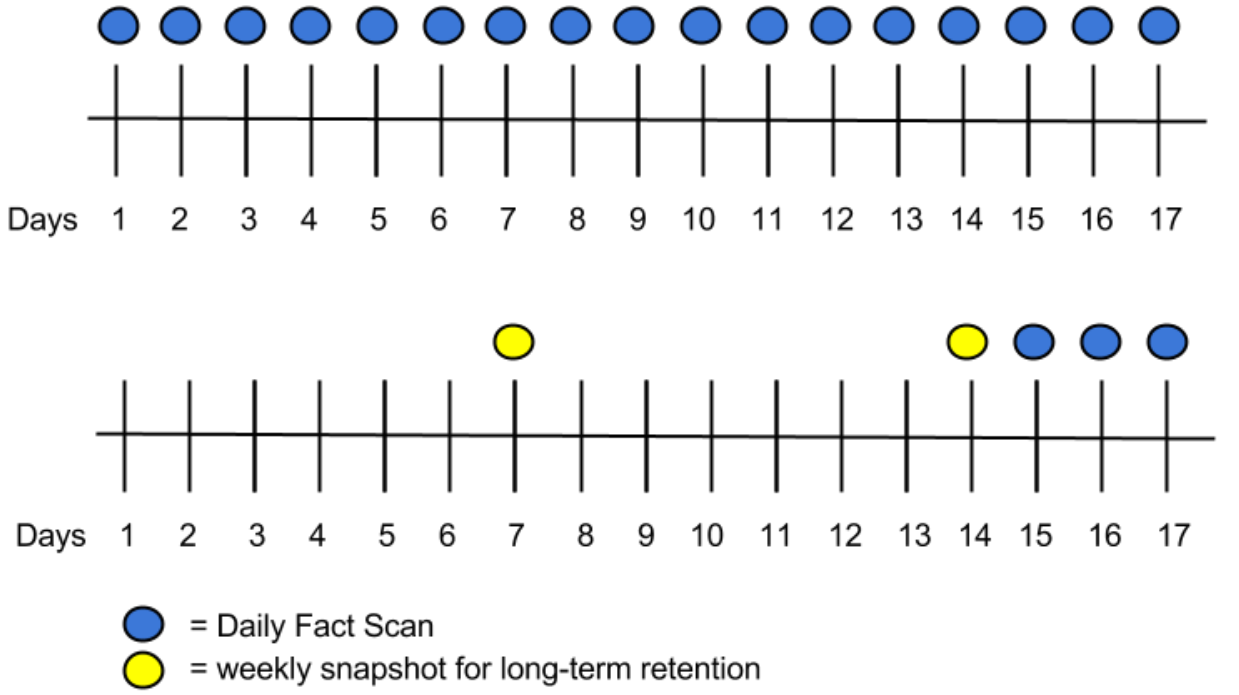


Select the **time period** after which you want to remove old data as well as the **frequency** for snapshot retention.

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.

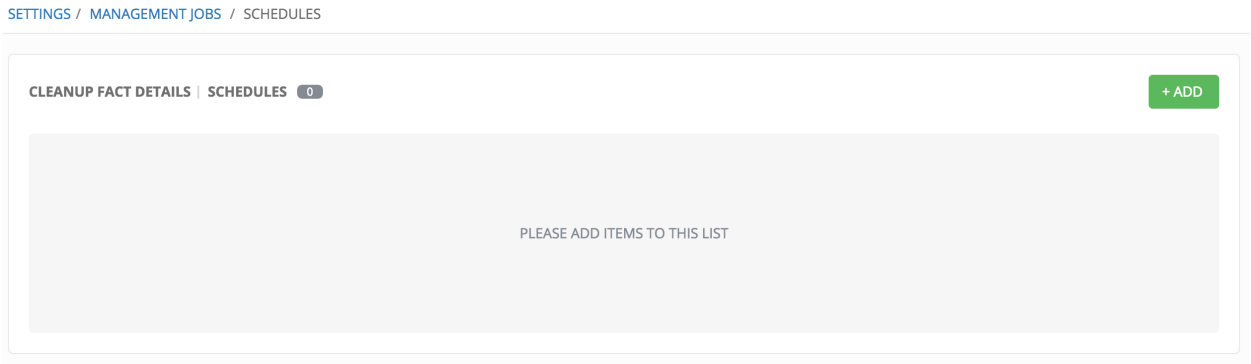
To help clarify this purge and retention schedule, consider the following timeline:



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.

4.2.1 Scheduling

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



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.)

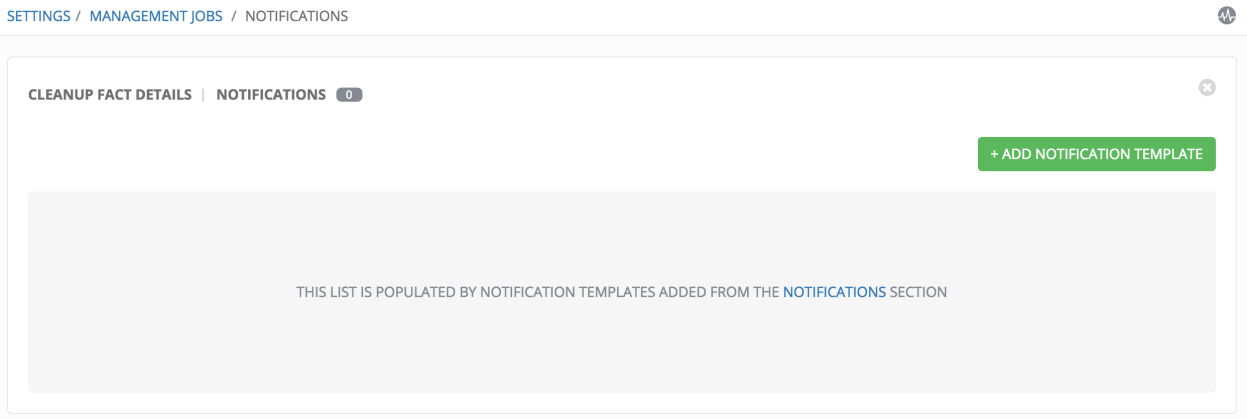
The **Details** tab displays a description of the schedule and a list of the scheduled occurrences in the selected Local Time Zone.

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.

4.2.2 Notifications

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

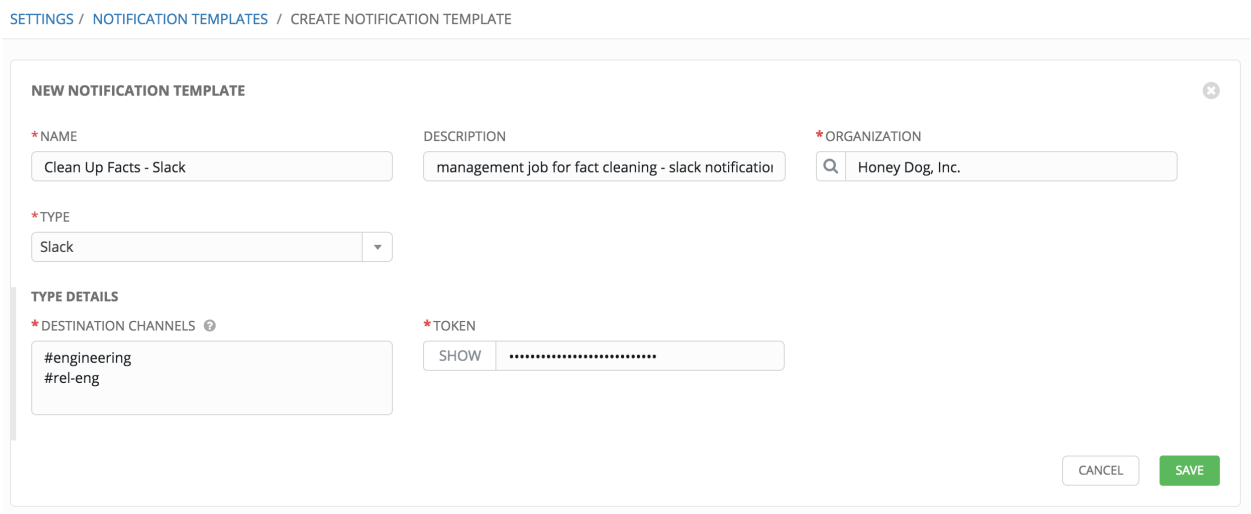
You can also access notifications through the Settings ( ) menu.



Click the

button to create a new notification. Notification types include:

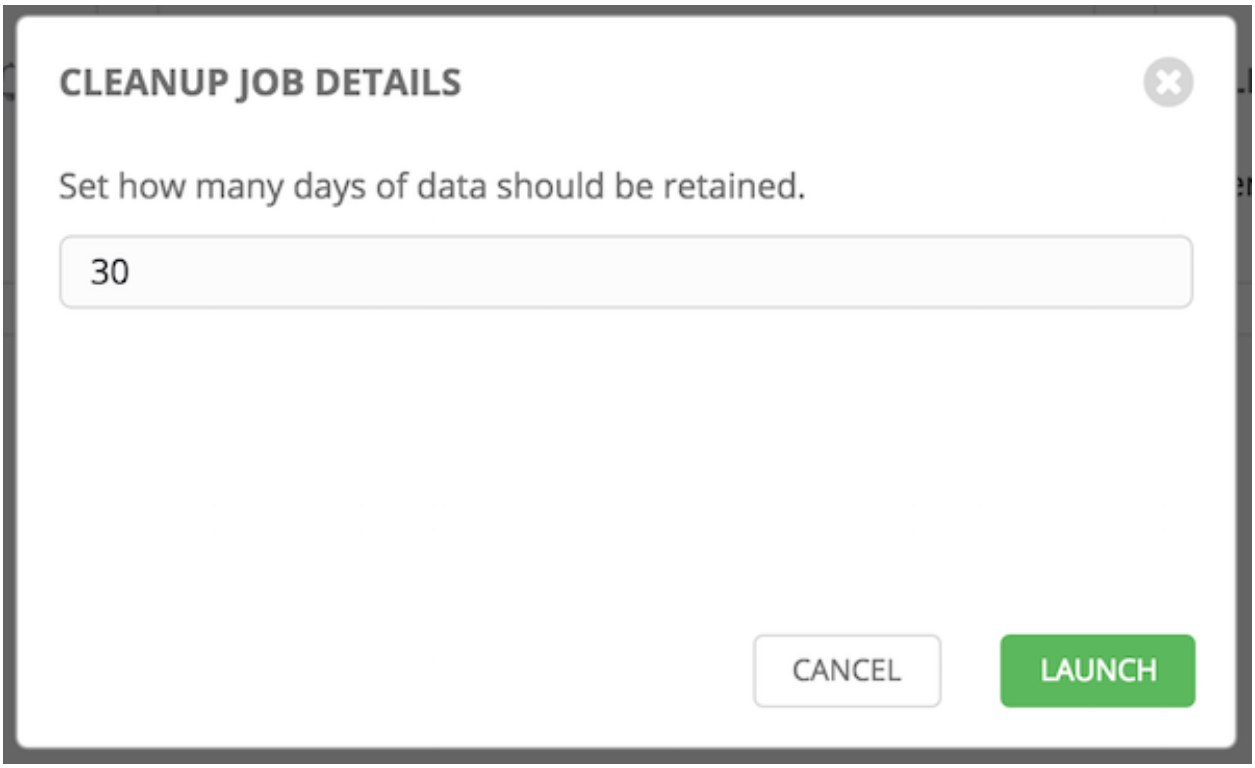
- Email
- Slack
- Twilio
- PagerDuty
- HipChat
- Webhook
- IRC



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

4.3 Removing Old Job History

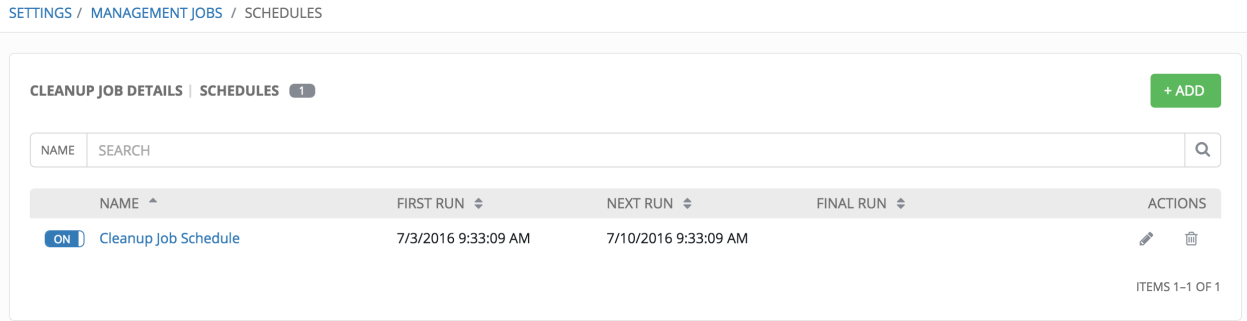
To remove job history older than a specified number of days, click on the button beside **Cleanup Job Details**.



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

4.3.1 Scheduling

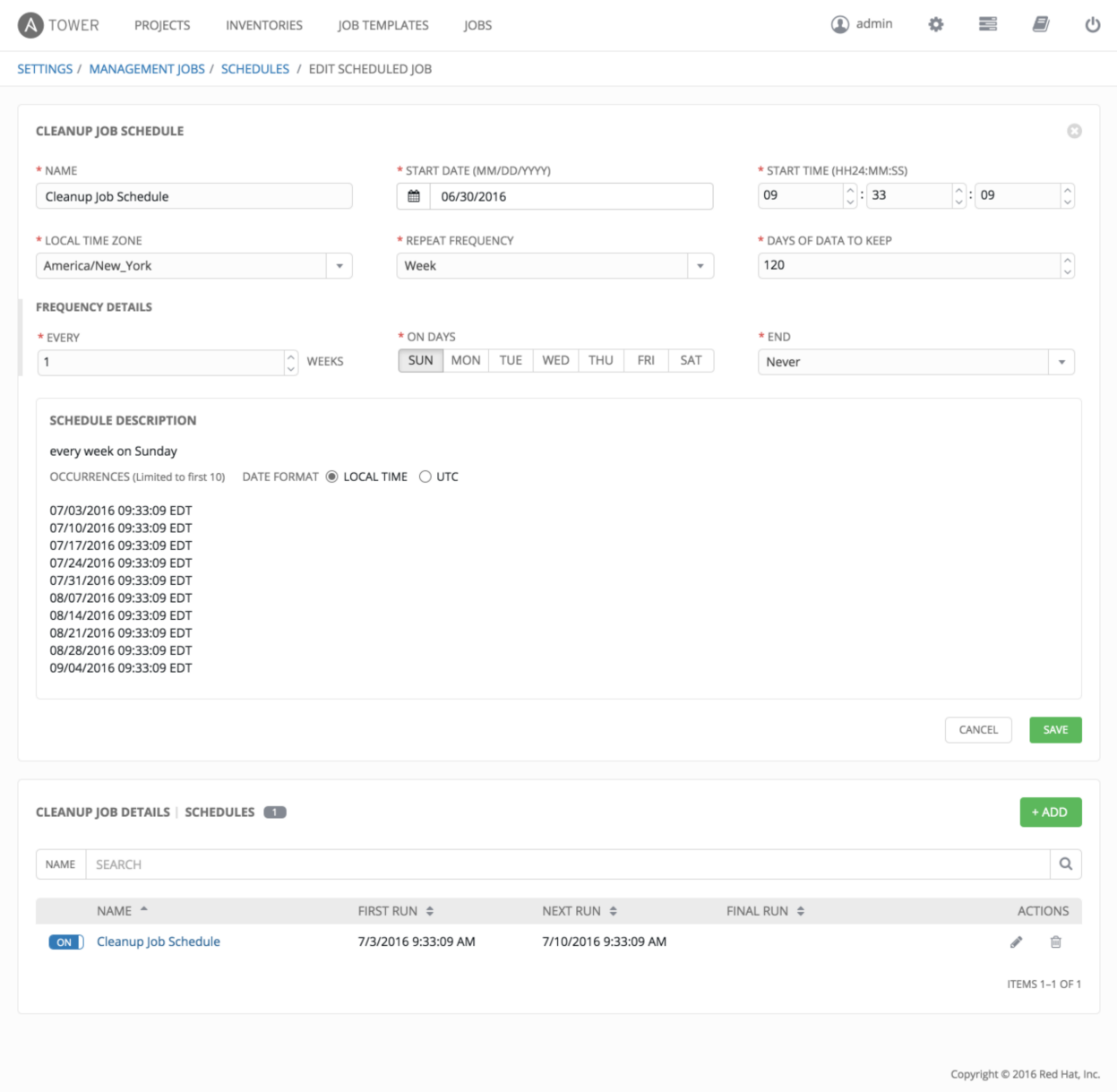
To review or set a schedule for cleaning up job history, click on the button.



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

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.

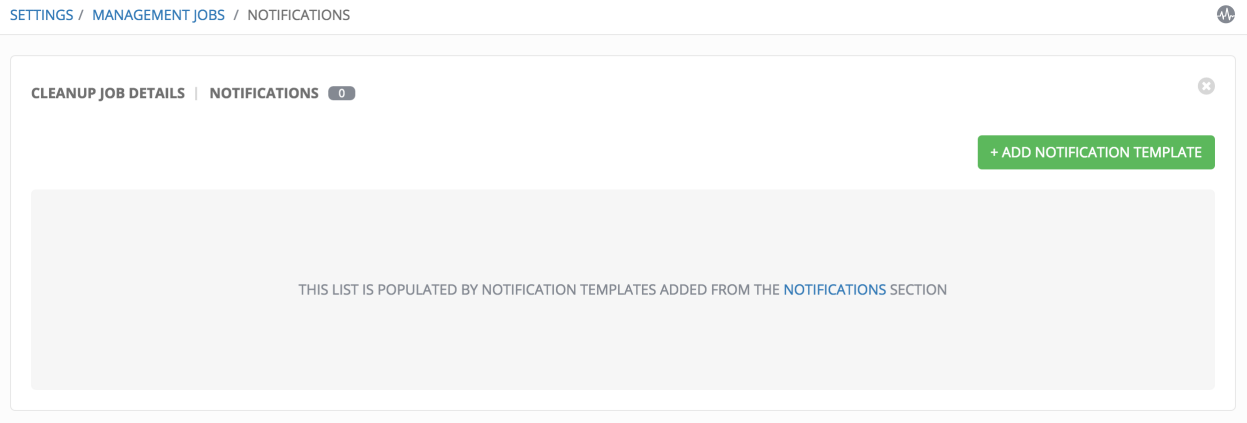
Note: Jobs are scheduled in UTC. Repeating jobs that runs at a specific time of day may move relative to a local

timezone when Daylight Saving Time shifts occur.

4.3.2 Notifications

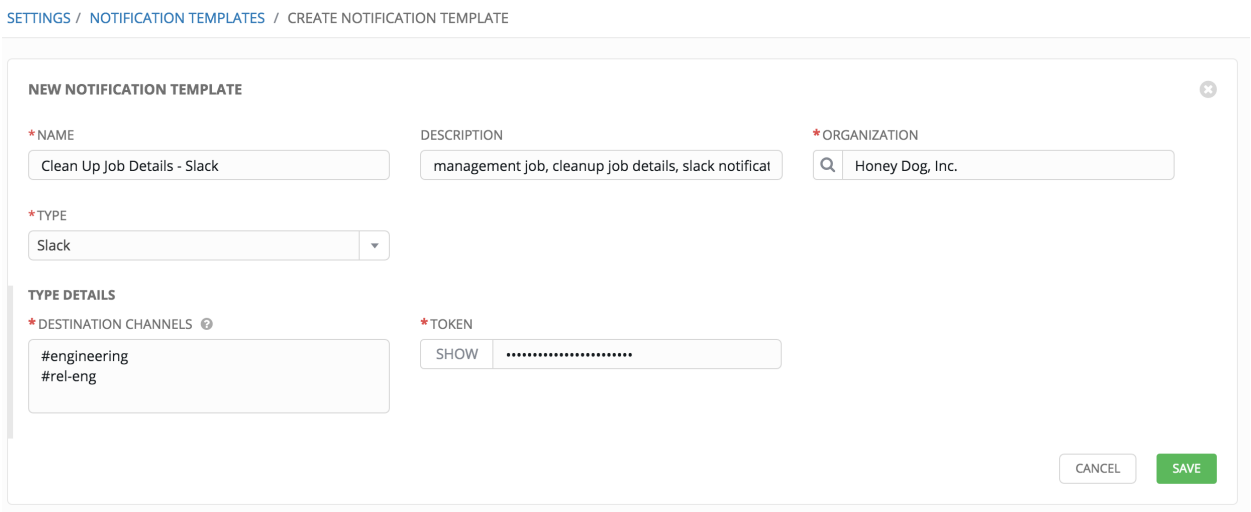
To set or review notifications associated with this management job, click the Configure Notifications ( ) button.

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



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

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



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

CHAPTER

FIVE

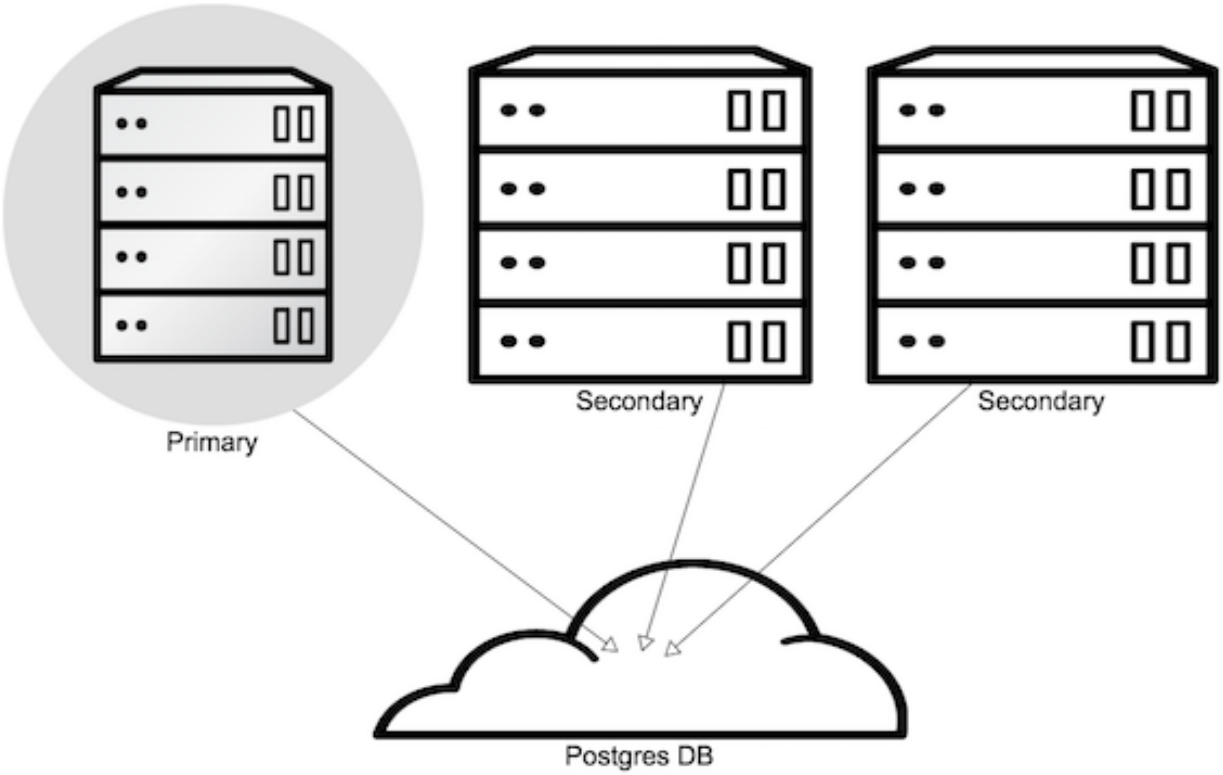
REDUNDANCY WITH TOWER

Note: Support for setting up a redundant environment is only available to those with Enterprise-level licenses.

Note: Prior to the Ansible Tower 2.4.5 release, the term High Availability had been used to describe Tower's redundancy offering. As it is not a true HA solution, references to High Availability, highly-available, and HA have been removed in favor of using the terms redundancy and redundant, to help eliminate further confusion.

Tower installations can occur in a Redundant (active/passive) configuration. In this configuration, Tower runs with a single active node, called the Primary instance, and any number of inactive nodes, called Secondary instances. Tower Redundancy runs as a "passive standby" solution and jobs on Secondary instances must first have that instance promoted to Primary before jobs can be started (which is not an automatic process and must be user triggered).

Secondary instances can become Primary at any time, with certain caveats. Running in a redundant setup requires any database that Tower uses to be external–Postgres must be installed on a machine that is not one of the primary or secondary tower nodes.



Tower's redundancy offers a standby Tower infrastructure that can become active in case of infrastructure failure–avoiding single points of failure. Redundancy is not meant to run in an active/active or multi-master mode, and is not a mechanism for horizontally scaling the Tower service. Remember that failover to a secondary instance is not automatic and must be user triggered.

For instructions on how to install into a redundant active/passive configuration, refer to the Ansible Tower Installation and Reference Guide.

Note: As of the release of Ansible Tower 3.0, the MongoDB is no longer packaged or used with Tower.

5.1 Setup Considerations

When creating a redundant deployment of Tower, consider the following factors:

- Tower is designed as a unit.

  Only those parts explicitly mentioned as being supported as external services are swappable for external versions of those services. Just as Tower does not support swapping out Django for Flask, Apache for lighttpd, or PostgreSQL for Oracle or MSSQL.

- Tower servers need isolation.

  If the primary and secondary Tower services share a physical host, a network, or potentially a data center, your infrastructure has a single point of failure. You should locate the Tower servers such that distribution occurs in a manner consistent with other services that you make available across your infrastructure. If your infrastructure is already using features such as Availability Zones in your cloud provider, having Tower distributed across Zones as well makes sense.

- The database require replication.

  If Tower runs in a redundancy setup, but the database is not run in a redundant or replicated mode, you still have a single point of failure for your Tower infrastructure. The Tower installer will not set up database replication; instead, it prompts for database connection details to an existing database (which needs replication).

  Choose a database replication strategy that is appropriate for your deployment.

  - For the general case of PostgreSQL, refer to the PostgreSQL documentation or the PostgreSQL Wiki.
  - For deployments using Amazon's RDS, refer to the Amazon documentation.

- Tower instances must maintain reasonable connections to the database.

  Tower both queries and writes to the database frequently; good locality between the Tower server and the database replicas is critical to ensure performance.

- Source Control is necessary.

  To use playbooks stored locally on the Tower server (rather than set to check out from source control), you must ensure synchronization between the primary and secondary Tower instances. Using playbooks in source control alleviates this problem.

  When using SCM Projects, a best practices approach is setting the *Update on Launch* flag on the job template. This ensures that checkouts occur each time the playbook launches and that newly promoted secondary instances have up-to-date copies of the project content. When a secondary instance is promoted, a project_update for all SCM managed projects in the database is triggered. This provides Tower with copies of all project playbooks.

- A consistent Tower hostname for clients and users.

  Between Tower users' habits, Tower provisioning callbacks, and Tower API integrations, keep the Tower hostname that users and clients use constant. In a redundant deployment, use a reverse proxy or a DNS CNAME. The CNAME is strongly preferred due to the websocket connection Tower uses for real-time output.

- When in a redundant setup, the remote Postgres version requirements are *Postgresql 9.4.x*.

  Postgresql 9.4.x is also required if Tower is running locally. With local setups, Tower handles the installation of these services. You should ensure that these are setup correctly if working in a remote setup.

  For help allowing remote access to your Postgresql server, refer to: http://www.thegeekstuff.com/2014/02/enable-remote-postgresql-connection/

For example, a redundant configuration for an infrastructure consisting of three datacenters places a Tower server and a replicated database in each datacenter. Clients accessing Tower use a DNS CNAME which points to the address of the current primary Tower instance.

For information determining size requirements for your setup, refer to Requirements in the *Ansible Tower Installation and Reference Guide*.

Note: Instances have been reported where reusing the external DB during subsequent redundancy installations causes installation failures.

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

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

5.2 Differences between Primary and Secondary Instances

The Tower service runs on both primary and secondary instances. The primary instance accepts requests or run jobs, while the secondary instances do not.

Connection attempts to the web interface or API of a secondary Tower server redirect to the primary Tower instance.

5.3 Post-Installation Changes to Primary Instances

When changing the configuration of a primary instance after installation, apply these changes to the secondary instances as well.

Examples of these changes would be:

- Updates to `/etc/tower/conf.d/ha.py`

  If you have configured LDAP or customized logging in `/etc/tower/conf.d/ldap.py`, you will need to reflect these changes in `/etc/tower/conf.d/ldap.py` on your secondary instances as well.

- Updating the Tower license

  Any secondary instance of Tower requires a valid license to run properly when promoted to a primary instance. Copy the license from the primary node at any time or install it via the normal license installation mechanism after the instance promotes to primary status.

Note: Users of older versions of Tower should update `/etc/tower/settings.py` instead of files within `/etc/tower/conf.d/`.

5.4 Examining the redundancy configuration of Tower

To see the redundancy configuration of Tower, you can query the **ping** endpoint of the Tower REST API. To do this via Tower's built in API browser, go to `https://<Tower server name>/api/v1/ping/`. You can go to this specific URL on either the primary or secondary nodes.

An example return from this API call would be (in JSON format):

```
HTTP 200 OK
Content-Type: application/json
Vary: Accept
Allow: GET, HEAD, OPTIONS
X-API-Time: 0.008s
{
    "instances": {
        "primary": "192.168.122.158",
        "secondaries": [
            "192.168.122.109",
            "192.168.122.26"
        ]
    },
    "ha": true,
    "role": "primary",
    "version": "2.1.4"
}
```

It contains the following fields.

- Instances
  - Primary: The primary Tower instance (hostname or IP address)
  - Secondaries: The secondary Tower instances (hostname or IP address)
- HA: Whether Tower is running in active/passive redundancy mode
- Role: Whether this specific instance is a primary or secondary
- Version: The Tower version in use

5.5 Promoting a Secondary Instance/Failover

To promote a secondary instance to be the new primary instance (also known as initiating failover), use the `tower_manage` command.

To make a running secondary node the primary node, log on to the desired new primary node and run the `update_instance` command of `tower_manage` as follows:

```
root@localhost:~$ tower-manage update_instance --primary
Successfully updated instance role (uuid="ec2dc2ac-7c4b-9b7e-b01f-0b7c30d0b0ab",
↪hostname="localhost",role="primary")
```

The current primary instance changes to be a secondary.

Note: Secondary nodes need a valid Tower license at `/etc/tower/license` to function as a proper primary instance. Copy the license from the primary node at any time or install it via the normal license installation mechanism after the instance promotes to primary status.

On failover, queued or running jobs in the database are marked as failed.

Tower does not attempt any health checks between primary or secondary nodes to do automatic failover in case of the loss of the primary node. You should use an external monitoring or heartbeat tool combined with `tower_manage` for these system health checks. Use of the `/ping` API endpoint could help.

Caution: If you are attempting to replicate across regions or other geographically separated locations, creating an HA environment will take a great deal of customization. Tower's HA is not meant to do full replication, as it is not written to be fully available globally. If attempting this type of replication, you should keep in mind the following:

- The `stdout` for your job history is not stored in the database but on the file system.
- Replicating the database does not provide you with all of your history, only highlights of important history moments.
- Projects folders should be reviewed as role dependencies may have placed the roles there or in other locations.
- Creating snapshots of VMs and backups may offer you more with less customization, allowing you to relaunch your Tower instance more easily (refer to *Backing Up and Restoring Tower* for more information).

5.6 Decommissioning Secondary instances

You cannot decommission a current primary Tower instance without first selecting a new primary.

If you need to decommission a secondary instance of Tower, log onto the secondary node and run the `remove_instance` command of `tower_manage` as follows:

```
root@localhost:~$ tower-manage remove_instance --hostname tower2.example.com
Instance removed (changed: True).
```

Replace `tower2.example.com` with the registered name (IP address or hostname) of the Tower instance you are removing from the list of secondaries.

You can then shutdown the Tower service on the decommissioned secondary node.

CHAPTER

SIX

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 does not work and is not supported. Tower requires the use of transparent proxies. If using ELB in AWS, ensure that "Proxy Protocol" is not enabled.

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']`

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

6.1 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

SEVEN

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-celery.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

EIGHT

THE *TOWER-MANAGE* UTILITY

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

8.1 Inventory Import

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

For help with `tower-manage`, run the following command: `tower-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`).

```
tower-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 do not already exist, but preserves any variables that were not sourced from the external data source.

```
tower-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.

8.2 Cleanup of old data

`tower-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.

- `tower-manage cleanup_jobs [--help]`

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

- `tower-manage cleanup-deleted [--help]`

This permanently deletes any deleted Tower objects that are older than a specified number of days.

- `tower-manage cleanup_activitystream [--help]`

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

8.3 HA management

Refer to the *Redundancy with Tower* section for details on the `tower-manage register_instance` and `tower-manage remove_instance` commands.

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

CHAPTER

NINE

LDAP INTEGRATION

Note: LDAP is an enterprise-level feature. You must have an active enterprise license before beginning the configuration process.

First, create a user in LDAP that has access to read the entire LDAP structure.

To test if you can make successful queries to the LDAP server, use the following command, 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.

This can be added to the Parameter `AUTH_LDAP_BIND_DN =` to specify the user that Tower uses to connect (Bind) to the LDAP server.

Note: LDAP and Active Directory integration is controlled by the settings defined in the `/etc/tower/conf.d/ldap.py` file. In versions of Tower older than 2.2, these settings lived in `/etc/tower/settings.py`.

The next parameter specifies which password to use for the Binding user:

```
AUTH_LDAP_BIND_PASSWORD = 'passme'
```

To define where to search for users while authenticating, use:

```
AUTH_LDAP_USER_SEARCH = LDAPSearch(
    'DC=WEBSITE,DC=COM',          # Base DN
    ldap.SCOPE_SUBTREE,           # SCOPE_BASE, SCOPE_ONELEVEL, SCOPE_SUBTREE
    '(sAMAccountName=%(user)s)',  # Query
)
```

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. For example, to query the AD LDAP using `ldapsearch` with a filter for the user, use something like:

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

Which returns something like:

```
# Administrator, Users, website.com
dn: CN=Administrator,CN=Users,DC=website,DC=com
objectClass: user
cn: Administrator
description: Built-in account for administering the computer/domain
distinguishedName: CN=Administrator,CN=Users,DC=website,DC=com
instanceType: 4
objectGUID:: lEphZCrQQUKedwWDg/KciQ==
accountExpires: 0
logonCount: 162
sAMAccountName: Administrator
sAMAccountType: 805306368
```

Here we see that name is stored in key `sAMAccountName`–hence the line becomes `'(sAMAccountName=%(user)s)'`.

Similarly, for OpenLDAP, the key is `uid`–hence the line becomes `'(uid=%(user)s)',`.

The next directives specifies the user attributes.

```
AUTH_LDAP_USER_ATTR_MAP = {
    'first_name': 'givenName',
    'last_name': 'sn',
    'email': 'mail',
}
```

The above example says that the users last name can be retrieved from the key `sn` in the `ldapsearch`. You can use the same LDAP query for the user to figure out what keys they are stored under.

The next directive specifies the where the groups should be searched and how to search them.

```
AUTH_LDAP_GROUP_SEARCH = LDAPSearch(
    'DC=website,DC=com',     # Base DN
    ldap.SCOPE_SUBTREE,      # SCOPE_BASE, SCOPE_ONELEVEL, SCOPE_SUBTREE
    '(objectClass=group)',   # Query
)
```

- 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.

You could make another `ldapsearch` and check in one group, which is the `objectclass` to which it belongs. For example:

```
# admin, grp, website.com
dn: CN=admin,OU=grp,DC=website,DC=com
objectClass: top
objectClass: group
```

```
cn: admin
member: CN=both,CN=Users,DC=website,DC=com
distinguishedName: CN=admin,OU=grp,DC=website,DC=com
```

The next directive specifies the type of group. Supported group types are listed here: http://pythonhosted.org/django-auth-ldap/groups.html#types-of-groups

```
AUTH_LDAP_GROUP_TYPE = ActiveDirectoryGroupType()
```

With these directives filled in and others commented, you now should be able to make a successful authentication with LDAP.

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

9.1 LDAPS

To enable secure LDAP communication with the LDAP server change the LDAP URL to LDAPS in the `AUTH_LDAP_SERVER_URI` directive. Make sure the server name in the URI matches the name in the certificate. Finally, add the server certificate to your Tower instance by adding the path which in CentOS is `/etc/openldap/ldap.conf` and the directive is `TLS_CACERT /etc/openldap/certs/cert.pem`.

To disable the certificate check, add the following lines to the `/etc/tower/conf.d/ldap.py` file:

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

9.2 Debugging

Debugging LDAP connections can be enabled by adding the below lines in the `/etc/tower/conf.d/ldap.py` file.

```
LOGGING['handlers']['syslog'] = {
    'level': 'DEBUG',
    'filters': ['require_debug_false'],
    'class': 'logging.handlers.SysLogHandler',
    'address': '/dev/log',
    'facility': 'local0',
    'formatter': 'simple',
}

LOGGING['loggers']['django_auth_ldap']['handlers'] = ['syslog']
LOGGING['loggers']['django_auth_ldap']['level'] = 'DEBUG'
```

9.3 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.

CHAPTER

TEN

SETTING UP AUTHENTICATION

Ansible Tower version 2.4.0 added authentication methods to 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.

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

For websites, such as Google or GitHub, that offer social functionality to users, social login is often implemented using the OAuth standard. OAuth is a secure authorization protocol which is commonly used in conjunction with 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 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.

10.1 Basic Authentication Settings:

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

```
AUTH_BASIC_ENABLED = False
```

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

10.2 Google OAuth2 Settings

Create a project at https://console.developers.google.com/ and obtain an OAuth2 key and secret for a web application. You will also need to provide the following callback URL for your application, replacing "tower.example.com" with the FQDN to your Tower server: `https://tower.example.com/sso/complete/google-oauth2/`

Ensure that the Google+ API is enabled.

Edit the `/etc/tower/conf.d/social_auth.py` file and enter in the appropriate values:

```
SOCIAL_AUTH_GOOGLE_OAUTH2_KEY = ''
SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET = ''
```

To restrict the domains who are allowed to login using Google OAuth2, uncomment the following line.

```
SOCIAL_AUTH_GOOGLE_OAUTH2_WHITELISTED_DOMAINS = ['example.com']
```

To only allow a single domain to authenticate using Google OAuth2, uncomment the following line; Google will not display any other accounts if the user is logged in with multiple Google accounts.

```
SOCIAL_AUTH_GOOGLE_OAUTH2_AUTH_EXTRA_ARGUMENTS = {'hd': 'example.com'}
```

Refer to the Python Social Auth documentation for advanced settings: https://python-social-auth.readthedocs.org/en/latest/backends/google.htmlgoogle-oauth2

Run `ansible-tower-service restart` on the Tower server to restart the instance and apply your changes.

10.3 Github OAuth2 Settings

Create a developer application at https://github.com/settings/developers and obtain an OAuth2 key (Client ID) and secret (Client Secret). You will also need to provide the following callback URL for your application, replacing "tower.example.com" with the FQDN to your Tower server: `https://tower.example.com/sso/complete/github/`

Edit the `/etc/tower/conf.d/social_auth.py` file and enter in the appropriate values:

```
SOCIAL_AUTH_GITHUB_KEY = ''
SOCIAL_AUTH_GITHUB_SECRET = ''
```

When defining authentication with either an organization or a team within an organization, you should use the specific organization and team settings. 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).

To limit users who can login to Tower, in this case, to only those in an organization or on a team within an organization, rather than using:

```
SOCIAL_AUTH_GITHUB_KEY
```

Use the following instead (depending on whether you are setting up for an organization or a team):

```
SOCIAL_AUTH_GITHUB_ORG_KEY

SOCIAL_AUTH_GITHUB_TEAM_KEY
```

To setup authentication for your organization, create an organization-owned application at `https://github.com/organizations/<yourorg>/settings/applications` and obtain an OAuth2 key (Client ID) and secret (Client Secret). Each key and secret must belong to a unique application and cannot be shared or reused between different social authentication backends. Provide the following callback URL for your application, replacing "tower.example.com" with the FQDN to your Tower server: `https://tower.example.com/sso/complete/github-org/`

```
SOCIAL_AUTH_GITHUB_ORG_KEY = ''
SOCIAL_AUTH_GITHUB_ORG_SECRET = ''
SOCIAL_AUTH_GITHUB_ORG_NAME = ''
```

To setup authentication for your team, create a team-owned application at `https://github.com/organizations/<yourorg>/settings/applications` and obtain an OAuth2 key (Client ID) and secret

(Client Secret). Each key and secret must belong to a unique application and cannot be shared or reused between different social authentication backends. You will also need to provide the following callback URL for your application, replacing "tower.example.com" with the FQDN to your Tower server: `https://tower.example.com/sso/complete/github-team/`

Find the numeric team ID using the Github API: http://fabian-kostadinov.github.io/2015/01/16/how-to-find-a-github-team-id/

```
SOCIAL_AUTH_GITHUB_TEAM_KEY = ''
SOCIAL_AUTH_GITHUB_TEAM_SECRET = ''
SOCIAL_AUTH_GITHUB_TEAM_ID = ''
```

Refer to Python Social Auth documentation for advanced settings: https://python-social-auth.readthedocs.org/en/latest/backends/github.html

Run `ansible-tower-service restart` on the Tower server to restart the instance and apply your changes.

10.4 SAML Authentication Settings

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

To setup SAML authentication, edit the `/etc/tower/conf.d/social_auth.py` file and enter in the appropriate values.

Set the `SOCIAL_AUTH_SAML_SP_ENTITY_ID` to a URL for a domain name you own (does not need to be a valid URL as this value is only used as a unique ID).

```
SOCIAL_AUTH_SAML_SP_ENTITY_ID = 'https://tower.example.com'
```

Create a keypair for Tower to use as a service provider (SP) and include the certificate and private key contents:

```
SOCIAL_AUTH_SAML_SP_PUBLIC_CERT = ''
SOCIAL_AUTH_SAML_SP_PRIVATE_KEY = ''
```

As an example for public certs:

```
SOCIAL_AUTH_SAML_SP_PUBLIC_CERT = '''

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

As an example for private keys:

```
SOCIAL_AUTH_SAML_SP_PRIVATE_KEY = '''
-----BEGIN PRIVATE KEY--
... key text ...
-----END PRIVATE KEY----
'''
```

Configure the following settings with information about your application and contact information:

```
SOCIAL_AUTH_SAML_ORG_INFO = {
    'en-US': {
        'name': 'example',
```

```
        'displayname': 'Example',
        'url': 'http://www.example.com',
    },
}
SOCIAL_AUTH_SAML_TECHNICAL_CONTACT = {
    'givenName': 'Some User',
    'emailAddress': 'suser@example.com',
}
SOCIAL_AUTH_SAML_SUPPORT_CONTACT = {
    'givenName': 'Some User',
    'emailAddress': 'suser@example.com',
}
```

Configure the entity ID, SSO URL and certificate for each identity provider (IdP) in use. 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). Attribute names may be overridden for each IdP as shown below.

```
SOCIAL_AUTH_SAML_ENABLED_IDPS = {
    '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',
    },
}
```

Your SAML identity provider (IdP) may provide a certificate to put into the x509cert field. You can take that certificate (assuming it is base64 encoded), remove the BEGIN and END lines, concatenate all of the remaining lines into one (making sure there are no spaces), and put that single long line, string into the field.

Run `ansible-tower-service restart` on the Tower server to restart the instance and apply your changes.

Once configuration is complete, you must register your SP with each IdP. Provide the entity ID and the following callback URL for your application, replacing "tower.example.com" with the FQDN to your Tower server: `https://tower.example.com/sso/complete/saml/`

If your IdP allows uploading an XML metadata file, you can download one from your Tower installation customized with the settings above: `https://tower.example.com/sso/metadata/saml/`

10.5 RADIUS Authentication Settings

Note: RADIUS 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. If backed by Active Directory, user access can be configured for RADIUS.

Edit the `/etc/tower/conf.d/radius.py` file and enter in the appropriate RADIUS server settings (skipped when RADIUS_SERVER is blank):

```
RADIUS_SERVER = ''
RADIUS_PORT = 1812
RADIUS_SECRET = ''
```

Run `ansible-tower-service restart` on the Tower server to restart the instance and apply your changes.

10.6 Using LDAP with Tower

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

Administrators use LDAP as a source for 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.

LDAP integration for Tower is configured in the file `/etc/tower/conf.d/ldap.py`. No configuration is accessible via the Tower user interface. Review the comments in that file for information on LDAP configuration and contact Ansible via the Red Hat Customer portal at https://access.redhat.com/ if you need further assistance.

Note: Users of older versions of Tower (prior to Tower version 2.3) should update `/etc/tower/settings.py` instead of files within `/etc/tower/conf.d/`.

10.6.1 Enabling Logging for LDAP

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

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

10.7 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 social authentication will automatically be added as admins of the organization.
- If False, no social 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. Compiled regular expressions may also be used instead of string literals.

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.

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

remove_users: True/False. Defaults to True.

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

```
SOCIAL_AUTH_ORGANIZATION_MAP = {
 Add all users to the default organization.
'Default': {
    'users': True,
},
'Test Org': {
    'admins': ['admin@example.com'],
    'users': True,
},
'Test Org 2': {
    'admins': ['admin@example.com', re.compile(r'^tower-[^@]+*?@.*$],
    'users': re.compile(r'^[^@].*?@example\.com$'),
},
}
```

Organization mappings may be specified separately for each social 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. Compiled regular expressions may also be used instead of string literals.

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': ['re.compile(r'^[^@]+?@test\.example\.com$')'],
    'remove': True,
},
'Other Team': {
    'organization': 'Test Org 2',
    'users': re.compile(r'^[^@]+?@test2\.example\.com$'),
    'remove': False,
},
}
```

Team mappings may be specified separately for each social 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

ELEVEN

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.

Create an API settings file (`/etc/tower/conf.d/session.py`) with the appropriately defined time variable:

```
AUTH_TOKEN_EXPIRATION = <seconds>  # default 1800
```

Create a `local_settings.json` file in `/var/lib/awx/public/static/local_settings.json` with any necessary settings.

The change from using a `local_config.js` file, which would overwrite all settings in the `config.js` file, to using a `local_settings.json` file, which only overwrites specific settings in the `config.js` file, was introduced in Ansible Tower version 2.4.

Note: When including a `local_settings.json` file with specifically configured variables, it will overwrite specific settings in the `config.js` file.

Tower is designed to look for the `config.js` file first, ensuring all preset configuration properties are set properly. Once loaded, Tower looks for a file called `local_settings.json` and checks to see which, if any, settings it should overwrite from `config.js`. Users can now specify only the properties they want to change.

Note: If you are using the `local_settings.js` file to configure some of your settings for the UI, you must switch to using the new `local_settings.json` file. If you do not, your Tower instance will be loaded with the default settings. Custom settings will not appear until you switch to using the new `local_settings.json` file.

For example, to turn the console debugger on, `local_settings.json` could contain the following object (no additional variables are needed):

```
{ "debug_mode" : true}
```

The variables you can use within the `local_settings.json` file are as follows:

- `tooltip_delay: {show: 500, hide: 100}` – Default number of milliseconds to delay displaying/hiding tooltips
- `debug_mode: false` – Enable console logging messages
- `password_length: 8` – Minimum user password length. Set to 0 to not set a limit

- `password_hasLowercase: true` – Requires a lowercase letter in the password
- `password_hasUppercase: false` – Requires an uppercase letter in the password
- `password_hasNumber: true` – Requires a number in the password
- `password_hasSymbol: false` – Requires one of these symbols to be in the password: -!$%^&*()_+|~=‘{}[]:”;’<>?,./
- `variable_edit_modes: {yaml, json}` – Options passed to ControlMirror for editing YAML/JSON variables (see below)

```
variable_edit_modes: {
           yaml: {
               mode:"text/x-yaml",
               matchBrackets: true,
               autoCloseBrackets: true,
               styleActiveLine: true,
               lineNumbers: true,
               gutters: ["CodeMirror-lint-markers"],
               lint: true
           },
           json: {
               mode: "application/json",
               styleActiveLine: true,
               matchBrackets: true,
               autoCloseBrackets: true,
               lineNumbers: true,
               gutters: ["CodeMirror-lint-markers"],
               lint: true
           }
       }
```

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

TWELVE

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
pip install kerberos
```

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 shold 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.

12.1 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, the password is unnecessary and the field can be left blank. Instead, you must `kinit` a ticket at the commandline as the `awx` user and have all the other inventory variables specified at the group level.

Kerberos needs the ticket created from `kinit` for authentication. Do not do this as the root user. Change to the `awx` user before you perform the `kinit`.

12.2 Working with Kerberos Tickets

Kerberos tickets are generated every 24 hours, as the default lifetime of a ticket is 24 hours. If you need to change this, edit the `/etc/krb.conf` file.

Another approach is to use `cron` to `kinit` the process every 24 hours. To automate this, you must generate a keytab file which stores the user password so that `kinit` will not prompt for the user password. Use the following steps to generate this keytab file and then get the kerberos ticket:

```
> ktutil
  ktutil:  addent -password -p username@WEBSITE.COM -k 1 -e aes256-cts-hmac-sha1-96
  provide password
  ktutil:  wkt username.keytab
  ktutil:  quit
```

Now, add the following command to `cron`:

```
kinit username@WEBSITE.COM -k -t username.keytab
```

Note: Make sure the system time is in sync between AD, Tower, and the clients.

Note: Client hostnames can looked up via DNS, both normally and reversed.

CHAPTER

THIRTEEN

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, edit the `sessions.py` file. 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

FOURTEEN

BACKING UP AND RESTORING TOWER

The ability to backup and restore your system(s) has been integrated into the Tower setup playbook, making it easy for you to backup and replicate your Tower instance as needed.

Note: The new backup/restore mechanism only works on systems which have been upgraded or installed as Ansible Tower 2.2. If you have an instance on an earlier version of Tower that you want to backup, for example, you must perform this process manually. For additional information, refer to: https://support.ansible.com/hc/en-us/articles/203295497-Tower-Manual-Backup-Restore

The Tower setup playbook is invoked as `setup.sh` from the path where you unpacked the Tower installer tarball. It uses the `tower_setup_conf.yml` and `inventory` files written by the Tower Installation Wizard. The setup script takes the following arguments for backing up and restoring:

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

14.1 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:

- To backup the configuration files, keys, and other relevant files, plus the database of the Tower installation.
- To restore the backed up files and data to a freshly installed and working second instance of Tower.

The `backup.yml` file looks like the following:

```
---
- hosts: primary
  gather_facts: yes
  roles:
    - backup
```

And is called within `setup.sh` as:

```
b)
    PLAYBOOK="backup.yml"
    TEMP_LOG_FILE="backup.log"
    OPTIONS="$OPTIONS --force-handlers"
    ;;
```

The `restore.yml` file looks like the following:

```
---
- hosts: primary
  gather_facts: yes
  roles:
    - restore
```

And is called within `setup.sh` as:

```
r)
    PLAYBOOK="restore.yml"
    TEMP_LOG_FILE="restore.log"
    BACKUP_FILE=`realpath $OPTARG`
    OPTIONS="$OPTIONS --force-handlers"
    ;;
```

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.

14.2 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.

CHAPTER

FIFTEEN

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.

To set up a custom logo, you must upload a file that you create called `custom_console_logo.png` into the `/var/lib/awx/public/static/assets/` directory. For the custom logo to look its best, use a `.png` file with a transparent background.

Next, either create a `/var/lib/awx/public/static/local_settings.json` file or add the following variable to the `local_settings.json` file if it already exists:

```
{ "custom_logo": true }
```

Setting the `custom_logo` variable to `false` (or not setting it at all), results in the appearance of the standard Ansible Tower logo.

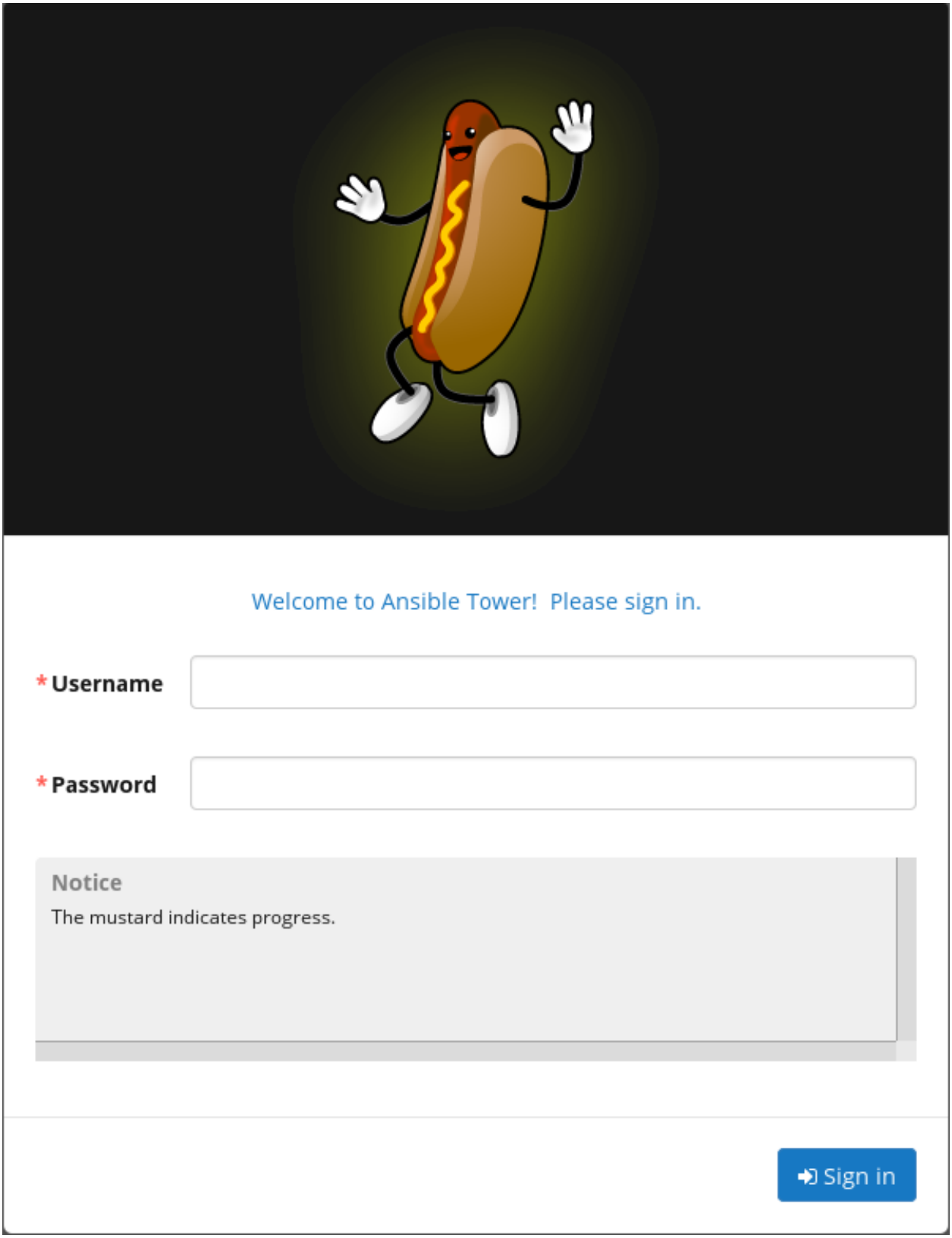
Note: If you cannot see the image in the Ansible Tower login dialog box after moving it into the `assets` directory, you must `chown` this file as the "awx" user, `chmod` this file as `755` for the correct permissions, and fix any SELinux labeling (if you are on Red Hat Enterprise Linux or CentOS, or another OS that uses SELinux). You can do this by using the `wget` or `scp` commands and directly accessing the `assets` directory (`/var/lib/awx/public/static/assets/`).

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 `"custom_login_info": "your notice content here"` to the object (`/var/lib/awx/public/static/local_settings.json`). Any content added must be in plain text, as custom HTML or other markup languages are not supported. If multiple paragraphs of text are needed, new lines (paragraphs) must be escaped as `\n` within the block of text, instead of being entered as a specific lines within the block of text in the `.json` file. Invalid `.json` files will error out if the syntax/formatting is incorrect.

For example, if you customized `/var/lib/awx/public/static/local_settings.json` as follows, and added the appropriate logo:

```
{  "custom_logo": true, "custom_login_info": "The mustard indicates progress." }
```

The Tower login dialog would look like this:



With regard to upgrades of Ansible Tower, both of the `local_settings.json` and `custom_console_logo.png` files should persist without any special work.

CHAPTER

SIXTEEN

TROUBLESHOOTING TOWER

16.1 Error logs

Tower server errors are logged in `/var/log/tower`. Supervisors logs can be found in `/var/log/supervisor/`. Apache 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/.

16.2 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.)

16.3 Changing the WebSockets port for live events

Ansible Tower uses port 8080 on the Tower server to stream live updates of playbook activity and other events to the client browser. If this port is already in use or is blocked by your firewall, you can reconfigure Tower to use a different port.

1. Create a `local_settings.json` file in `/var/lib/awx/public/static/local_settings.json`, add an entry for `websocket_port`, and set the value to the desired port, such as: `{"websocket_port": 8080}`.
2. Edit `/etc/tower/settings.py` and add a new line like the following (in this example, 8081 is your new desired port): `SOCKETIO_LISTEN_PORT=8081`
3. Make sure your firewall allows traffic through this port.
4. Restart Tower by running the admin utility script, `ansible-tower-service restart`.

Note that `local_settings.json` is removed when upgrading Tower to a new release. You must recreate and reapply the change on each upgrade of Tower.

16.4 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.

16.5 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`.

16.6 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/
```

16.7 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.

16.8 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.

16.9 Reusing an external HA database causes installations to fail

Instances have been reported where reusing the external DB during subsequent HA installations causes installation failures.

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

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

16.10 PRoot functionality and variables

The PRoot 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 PRoot settings in some cases. To fine tune your usage of PRoot, there are certain variables that can be set:

```
# Enable proot support for running jobs (playbook runs only).
AWX_PROOT_ENABLED = False

# Command/path to proot.
AWX_PROOT_CMD = 'proot'

# Additional paths to hide from jobs using proot.
AWX_PROOT_HIDE_PATHS = []

# Additional paths to show for jobs using proot.
AWX_PROOT_SHOW_PATHS = []
```

To customize your PRoot settings, navigate to `the /etc/tower/settings.py` file. Once your changes have been saved, restart services with the `ansible-tower-service restart` command.

16.11 Private EC2 VPC Instances in Tower Inventory

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

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

```
vpc_destination_variable: private_ip_address
```

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

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

16.12 Troubleshooting "Error: provided hosts list is empty"

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

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

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

CHAPTER

SEVENTEEN

TOWER TIPS AND TRICKS

17.1 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.

17.2 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/v1/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/v1/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: 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.

17.3 `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
```

17.4 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:

```
tower-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.

17.5 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:

```
tower-manage createsuperuser
```

17.6 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 by navigating to the `/etc/tower/settings.py` file, setting `AWX_PROOT_ENABLED=False`, then restarting services with the `ansible-tower-service restart` command.

17.7 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/v1/jobs/<job_id>/job_events/
```

17.8 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. You can also install your own copy in `~/.ansible.cfg` or keep a copy in a directory relative to your playbook named as `ansible.cfg`.

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.

17.9 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

17.10 Using virtualenv with Ansible Tower

Ansible Tower 3.0 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.

Note: For more information on virtualenv, see Virtual Environments

17.10.1 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
```

17.10.2 Tower virtualenv under Red Hat Enterprise Linux 6 and derivatives

Red Hat Enterprise Linux 6 and derivatives (including CentOS 6 and Oracle Linux 6) use Python version 2.6. However, Tower 3.0 requires several components (most notably Django 1.8) that require Python 2.7, so the Python 2.7 software collection is installed with Ansible Tower on these platforms. Note that this is only used to run Tower, not Ansible. Ansible still runs under the system Python 2.6. As a result, the two virtualenvs under `/var/lib/awx/venv` will each target a different Python interpreter.

17.11 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.

17.12 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/v1/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/v1/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.

17.13 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.

17.14 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.

17.15 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.

17.16 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

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 the callback 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
```

17.17 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.

17.18 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 `tower-manage` command:

```
tower-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:

```
tower-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:

```
tower-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
```

Your inventory file is most likely in "[groupname:vars]" structure. At this time, the inventory importer tool does not support this format. For each of the groups that has vars attached, move those groups into a group_vars file.

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

EIGHTEEN

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.

18.1 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.

18.2 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

18.3 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/

18.4 Configuration

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

18.4.1 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.

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
```

18.4.2 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
```

18.4.3 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

18.4.4 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
```

18.4.5 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

18.4.6 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‘‘, to true. To allow it for a single command, 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

NINETEEN

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.

To opt out, navigate to the `/etc/tower/` directory and set the following in `settings.py`:

```
PENDO_TRACKING_STATE = 'off'
```

Once set, you must restart your instance of Tower using the `ansible-tower-service restart` command, re-authenticate, and force-reload your browser session.

To re-enable data collection, navigate to the `/etc/tower/` directory and set the following in `settings.py`:

```
PENDO_TRACKING_STATE = 'detailed'
```

Once set, you must restart your instance of Tower using the `ansible-tower-service restart` command, re-authenticate, and force-reload your browser session.

To enable data collection without your specific user data, navigate to the `/etc/tower/` directory and set the following in `settings.py`:

```
PENDO_TRACKING_STATE = 'anonymous'
```

Once set, you must restart your instance of Tower using the `ansible-tower-service restart` command, re-authenticate, and force-reload your browser session.

CHAPTER

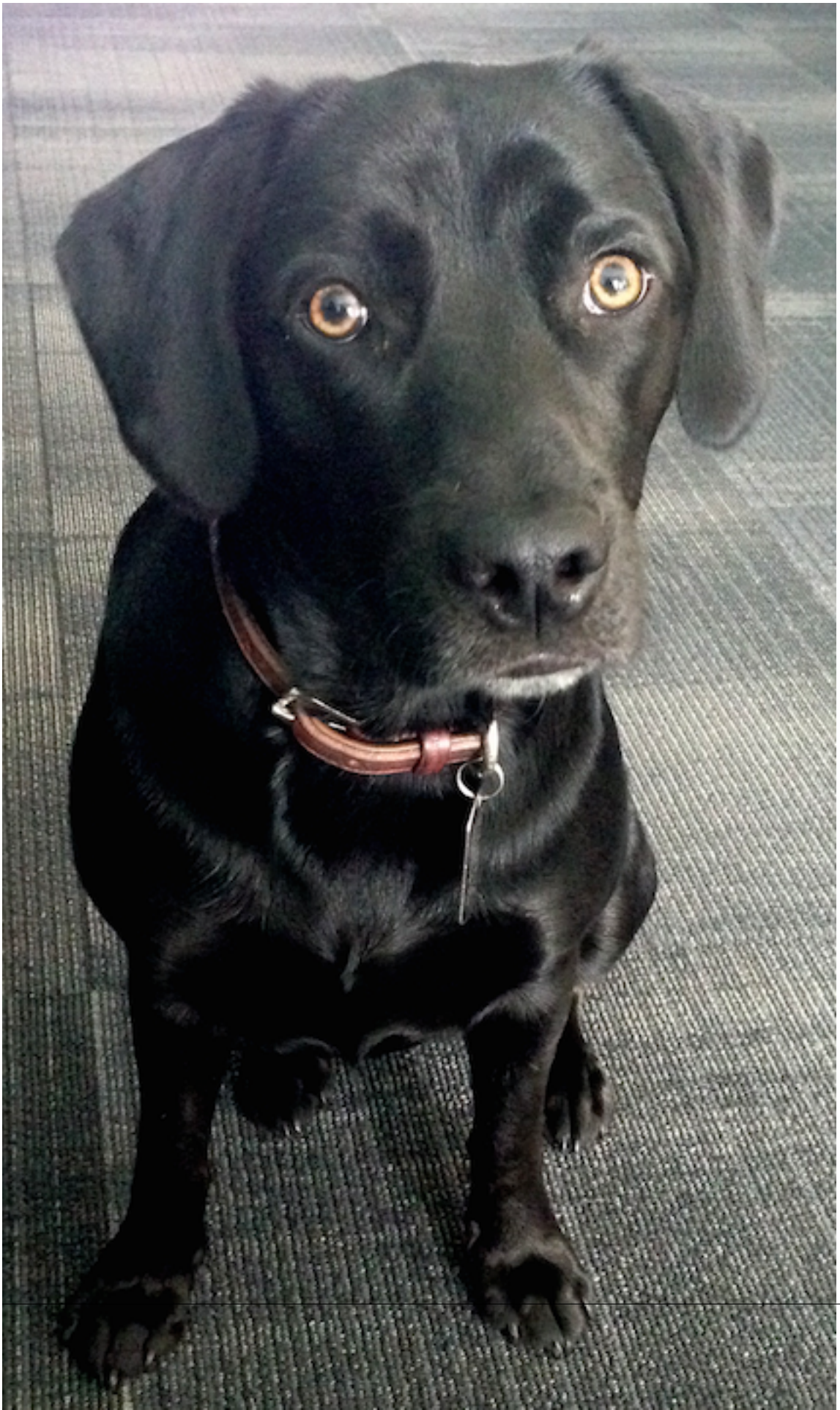
TWENTY

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

TWENTYONE

INDEX

- genindex

CHAPTER

TWENTYTWO

COPYRIGHT © 2016 RED HAT, INC.

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

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

Third Party Rights

Ubuntu and Canonical are registered trademarks of Canonical Ltd.

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

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

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

Rackspace trademarks, service marks, logos and domain names are either common-law trademarks/service marks or registered trademarks/service marks of Rackspace US, Inc., or its subsidiaries, and are protected by trademark and other laws in the United States and other countries.

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

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

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

Safari® is a registered trademark of Apple, Inc.

Firefox® is a registered trademark of the Mozilla Foundation.

All other trademarks are the property of their respective owners.

INDEX

Symbols

: API
launching a Job Template, 56

A

Active Directory (AD)
Kerberos, 44
activity stream cleanup management job, 9
admin creation
commandline, 58
tips, 58
admin password
changing password, 58
admin password change
tips, 58
admin utility script, 5
Amazon RDS
redundancy, 22
analytics collection, 68
Ansible configuration file, 59
Ansible modules, unreleased
tips, 62
Ansible output for JSON commands, 59
Ansible, executing in a virtual enviroment, 60
ansible-tower script replacement, 5
ansible.cfg, 59
tips, 59
ansible_variables, viewing all
tips, 59
AUTH_BASIC_ENABLED
session limits, 47
AUTH_TOKEN_PER_USER
session limits, 47
authentication, 35
LDAP, 39
organization mapping, 39
RADIUS Authentication Settings, 38
SAML Service Provider, 37
team mapping, 39
authentication expiring, 42, 43
authentication timeout
changing the default, 42
troubleshooting, 42
authentication token, 42, 43

B

backups, 48
considerations, 49
playbooks, 48
basic authentication, 35
best practices, 56

C

callback plugins
tips, 62
change password
tower-manage, 58
changing password
admin password, 58
changing the default
authentication timeout, 42
cleaning old data, 9
cleanup activity stream
management jobs, 9
cleanup fact details
management jobs, 13
cleanup job history
management jobs, 16
command line interface
tips, 56
Tower CLI, 56
commandline
admin creation, 58
components
licenses, 4
configuration file configuration
tips, 59
configuration file location
tips, 59
curl
tips, 60
custom
logo, 50
custom inventory scripts, 6

custom logo, 50

D

data collection, 68
DEB files
    licenses, 4
debugging
    LDAP, 33
dynamic inventory and instance filtering
    tips, 61
dynamic inventory and private IPs
    tips, 61

E

EC2
    VPC instances, 54
EC2 VPC instances
    tips, 61
    troubleshooting, 54
enterprise authentication, 35
error logs
    troubleshooting, 52
evaluation, 3
external HA database
    installation failure, 54

F

fact details cleanup management job, 13
features, 1
filtering instances
    tips, 61
functionality
    PRoot, 54

G

general help
    troubleshooting, 52
Github OAuth2
    social authentication, 36
Google OAuth2
    social authentication, 35

H

help, 52, 56
high availability, *see* other: redundancy
host connections
    troubleshooting, 52, 53
host/group vars import
    tips, 63
hostname configuration
    notifications, 60
hosts list
    troubleshooting, 55
hosts lists (empty), 55

I

importing host/group vars
    importing inventory, 63
importing inventory
    importing host/group vars, 63
init script replacement, 5
installation bundle
    licenses, 4
installation failure
    external HA database, 54
installation wizard
    playbook backup/restore arguments, 48
instance filtering
    tips, 61
inventory import
    tips, 63
inventory scripts
    custom, 6
    writing, 8

J

job cancellation
    troubleshooting, 53
job does not run
    troubleshooting, 53
job history cleanup management job, 16
Job Template drop-down list
    playbooks are not viewable, 53
JSON commands, Ansible output, 59
jump host
    ProxyCommand, 59
    tips, 59

K

Kerberos
    Active Directory (AD), 44
    user authentication, 44

L

launching a Job Template
    : API, 56
LDAP, 31, 39
    authentication, 39
    debugging, 33
    referrals, 33
    troubleshooting, 31
LDAP debugging
    troubleshooting, 33
LDAP referrals
    troubleshooting, 33
LDAPS, 33
    troubleshooting, 33
license, 1, 2

features, 4
nodes, 3
trial, 3
types, 3
license features, 1
licenses
components, 4
DEB files, 4
installation bundle, 4
RPM files, 4
live events
port changes, 52
troubleshooting, 52
log, 42, 43
logfiles, 28
login timeout, 42
logo
custom, 50

M

management jobs, 9
cleanup activity stream, 9
cleanup fact details, 13
cleanup job history, 16
modules, using unreleased
tips, 62

N

notifications
hostname configuration, 60

O

organization mapping, 39
authentication, 39

P

pending playbook
troubleshooting, 53
Pendo, 68
PENDO_TRACKING_STATE, 68
playbook setup
backup/restore arguments, 48
playbooks are not viewable
Job Template drop-down list, 53
playbooks not appearing
troubleshooting, 53
plugins, callback
tips, 62
port changes
live events, 52
postface, 69
Postgresql
redundancy, 22
primary instances
post-installation changes, 24
redundancy, 24
private IPs with dynamic inventory
tips, 61
PRoot
functionality, 54
troubleshooting, 53, 54
variables, 54
proxy support, 27
reverse proxy, 27
ProxyCommand
jump host, 59
tips, 59

R

RADIUS Authentication Settings
authentication, 38
rebranding, 50
redundancy, 21
Amazon RDS, 22
Postgresql, 22
primary instances, 24
secondary instances, 24
setup considerations, 22
viewing configuration, 24
referrals
LDAP, 33
removing old data, 9
restart Tower, 5
restorations, 48
considerations, 49
playbooks, 48
reverse proxy, 27
proxy support, 27
RPM files
licenses, 4

S

SAML Service Provider
authentication, 37
scripts, admin utility, 5
secondary instances
decommissioning, 26
failover, 25
promotion to primary, 25
redundancy, 24
session limits, 47
AUTH_BASIC_ENABLED, 47
AUTH_TOKEN_PER_USER, 47
session.py, 47
social authentication, 35
Github OAuth2, 36
Google OAuth2, 35
start Tower, 5

stop Tower, 5
super user creation
    tower-manage, 58
support, 1–3

T

team mapping, 39
    authentication, 39
timeout login, 42
tips, 56
    admin creation, 58
    admin password change, 58
    Ansible modules, unreleased, 62
    ansible.cfg, 59
    ansible_variables, viewing all, 59
    callback plugins, 62
    command line interface, 56
    configuration file configuration, 59
    configuration file location, 59
    curl, 60
    dynamic inventory and instance filtering, 61
    dynamic inventory and private IPs, 61
    EC2 VPC instances, 61
    filtering instances, 61
    host/group vars import, 63
    instance filtering, 61
    inventory import, 63
    jump host, 59
    modules, using unreleased, 62
    plugins, callback, 62
    private IPs with dynamic inventory, 61
    ProxyCommand, 59
    Tower CLI, 56
    unreleased modules, 62
    Windows connection, 62
    winrm, 62
tools
    tower-cli, 64
Tower admin utility script, 5
Tower CLI
    command line interface, 56
    tips, 56
tower-cli, 64
    capabilities, 64
    installation, 64
tower-manage, 25, 29
    change password, 58
    high availability management, 30
    inventory import, 29
    super user creation, 58
tower-manage, data cleanup, 29
trial, 3
troubleshooting, 52
    authentication timeout, 42
    EC2 VPC instances, 54
    error logs, 52
    general help, 52
    host connections, 52, 53
    hosts list, 55
    job cancellation, 53
    job does not run, 53
    LDAP, 31
    LDAP debugging, 33
    LDAP referrals, 33
    LDAPS, 33
    live events, 52
    pending playbook, 53
    playbooks not appearing, 53
    PRoot, 53, 54
    websockets, 52

U

unreleased modules
    tips, 62
updates, 3
usability data collection, 68
user authentication
    Kerberos, 44
user data tracking, 68

V

variables
    PRoot, 54
virtual enviroment, 60
VPC instances
    EC2, 54

W

websockets
    troubleshooting, 52
Windows connection
    tips, 62
winrm
    tips, 62