Ansible Tower Administration Guide

Release Ansible Tower 2.2.0

Ansible, Inc.

Jun 06, 2017

CONTENTS

Thank you for your interest in Ansible Tower, the open source IT orchestration engine. Whether sharing operations tasks with your team or integrating with Ansible through the Tower REST API, Tower provides powerful tools to make your automation life easier.

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 2.2.0.

Ansible Tower Version 2.2.0; July 14, 2015; https://access.redhat.com/

CHAPTER

ONE

TOWER SERVICES SCRIPT

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: Ansible Tower 2.2.0 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`. The new admin utility script, `ansible-tower-service`, should be used instead.

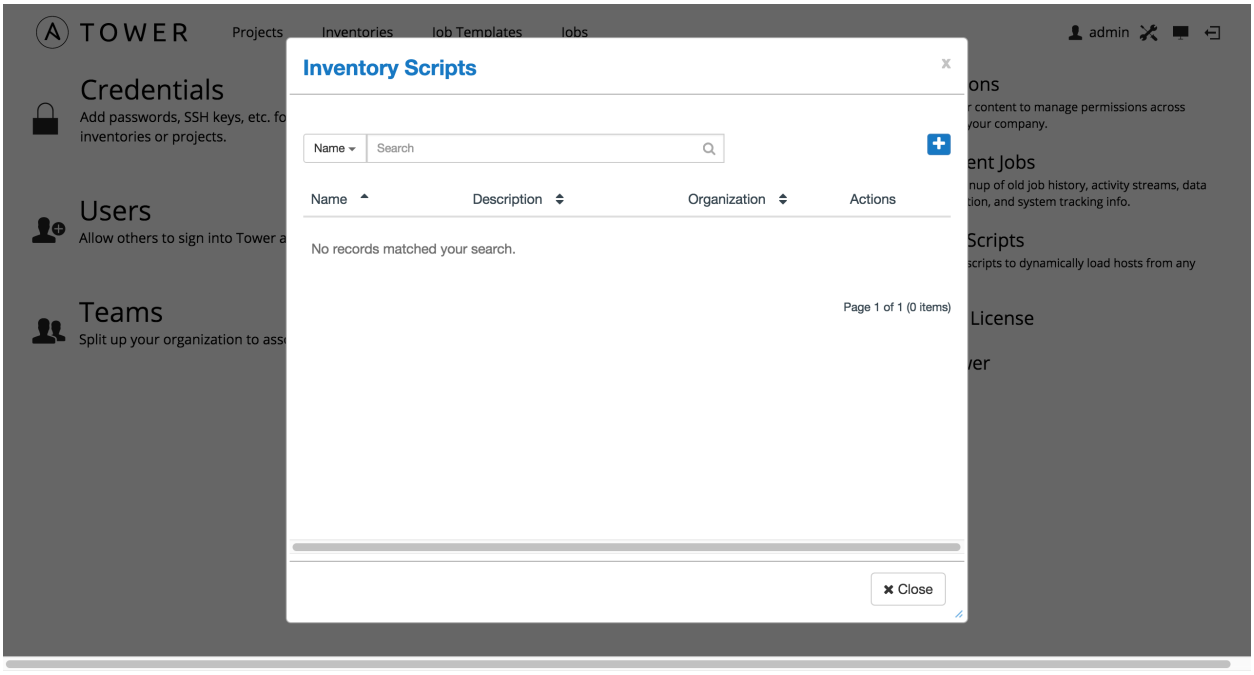
CHAPTER

TWO

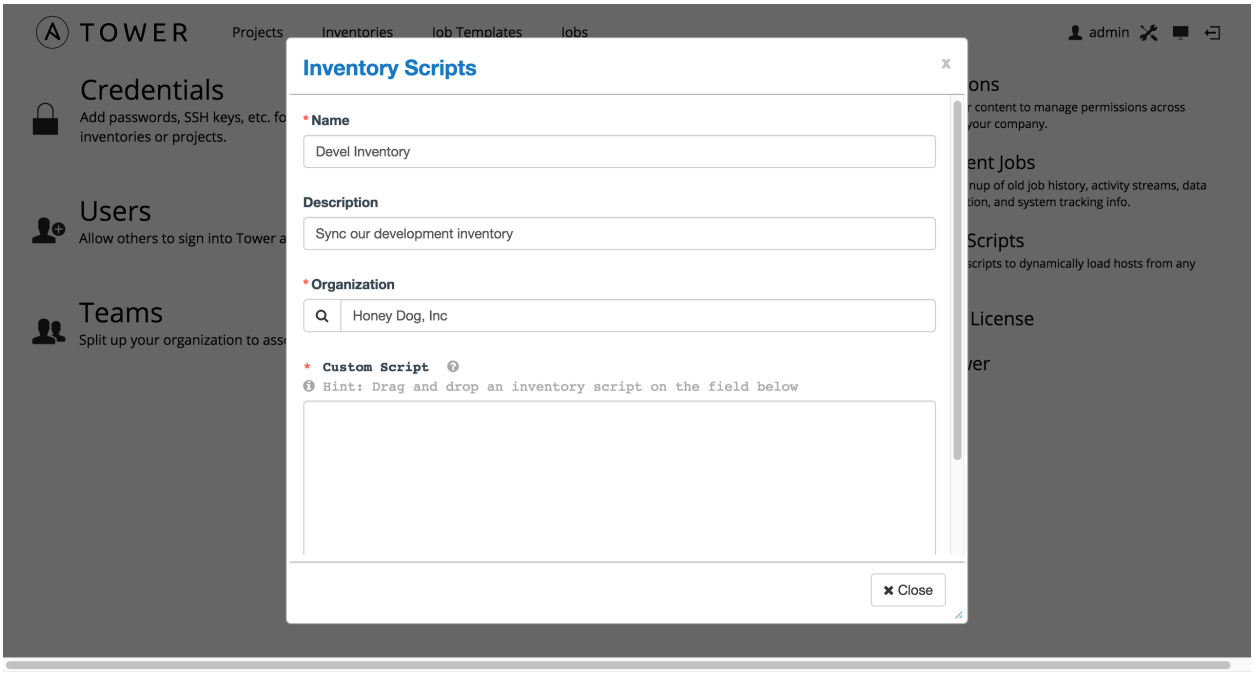
CUSTOM INVENTORY SCRIPTS

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.

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  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.

2.1 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`.

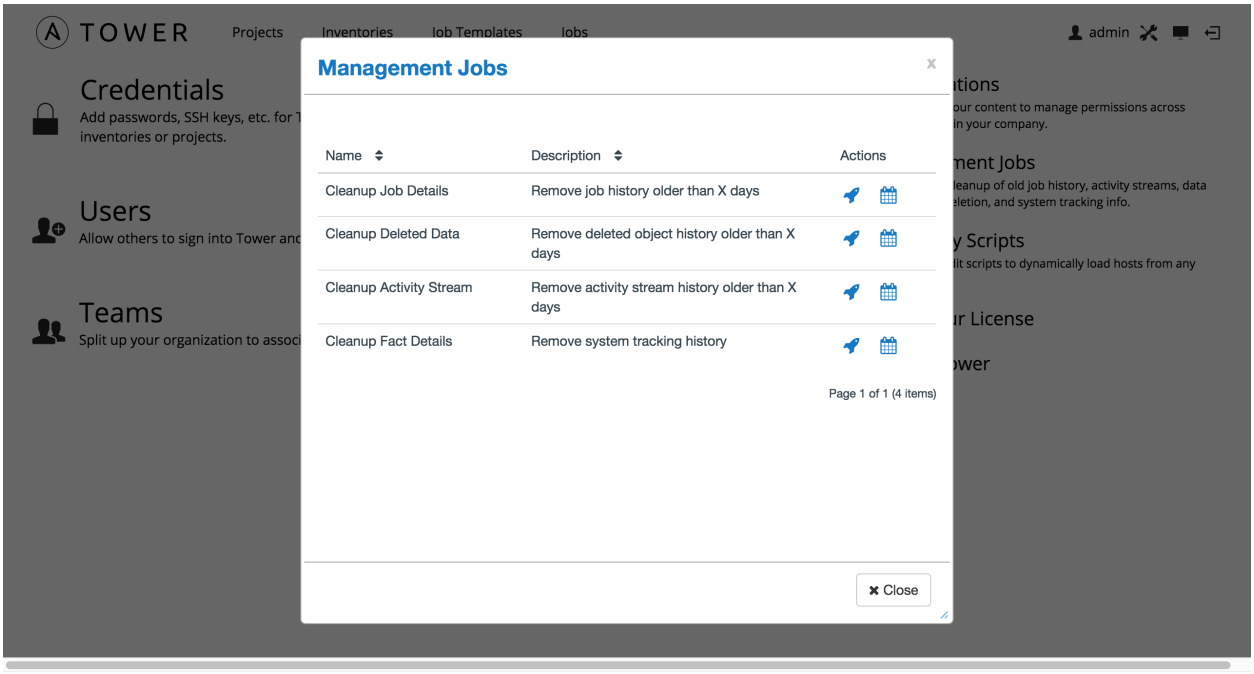
For more information on dynamic inventory scripts and how to write them, refer to the Intro to Dynamic Inventory and Developing Dynamic Inventory Sources sections of the Ansible documentation, or review the example dynamic inventory scripts on GitHub.

CHAPTER

THREE

MANAGEMENT JOBS

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

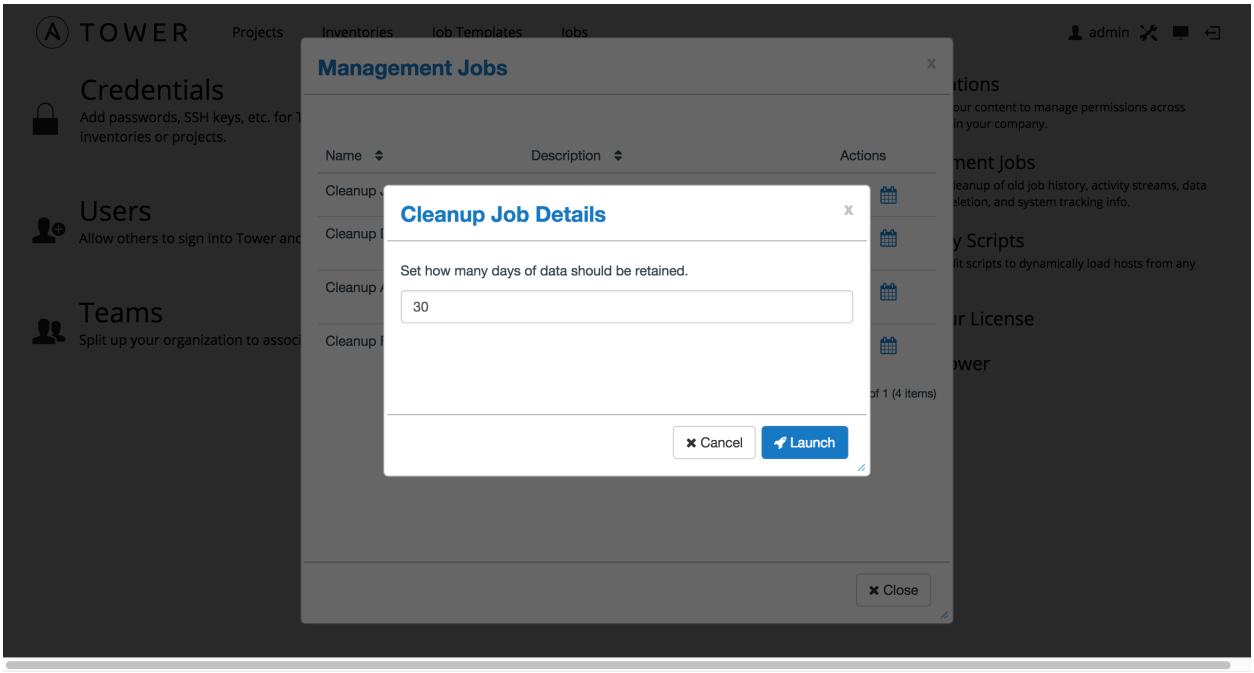


Several job types are available for you to schedule and launch:

- **Cleanup Job Details**: Remove job history older than a specified number of days
- **Cleanup Deleted Data**: Remove deleted object history older than a specified number days
- **Cleanup Activity Stream**: Remove activity stream history older than a specified number of days
- **Cleanup Fact Details**: Remove system tracking history

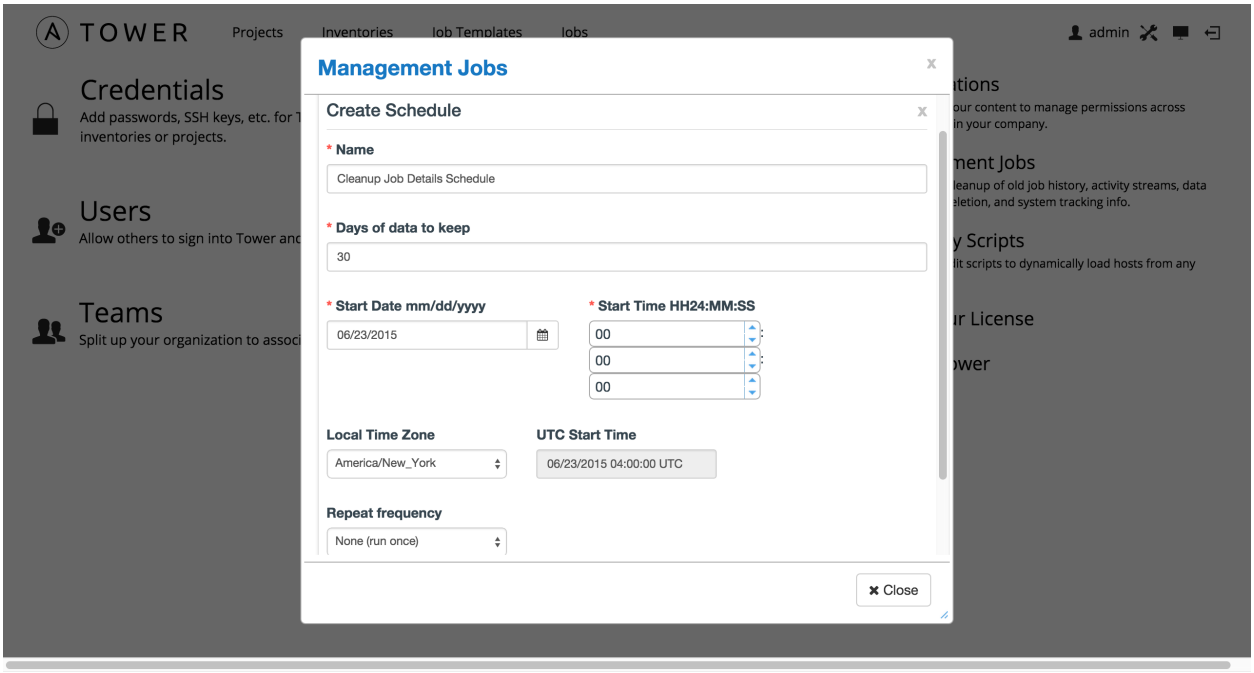
3.1 Removing Old Job History

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**.

To set a schedule for cleaning up old job history information, click on the  button.



Enter the appropriate details for:

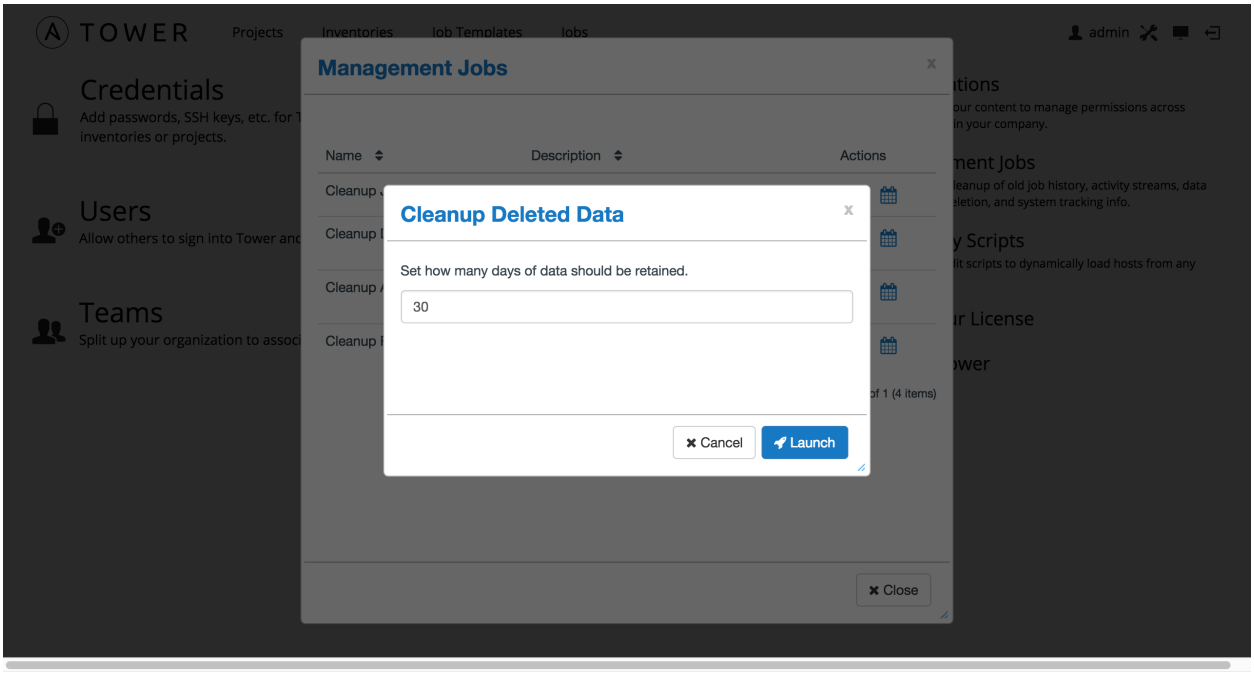
- **Name**: populated by default to match the management job type and can be left alone
- **Days of data to keep**: Enter the number of days of data you would like to retain.
- **Start Date**: Enter the date for which you want data cleanup to start (month/day/year format)
- **Start Time**: Enter the time to start data cleanup in hours, minutes, and seconds.

- **Local Time Zone**: Select your preferred time zone.
- **UTC Start Time**: Review the UTC Start time which is based on the Start Time and Local Time Zone you select.
- **Repeat Frequency**: Select how often you want the cleanup job to run. You can enter cleanups to occur once or every *X* minutes, hours, days, weeks, months, or years.

The **View Details** link at the bottom displays a description of the schedule and a list of the scheduled occurrences in the selected Local Time Zone.

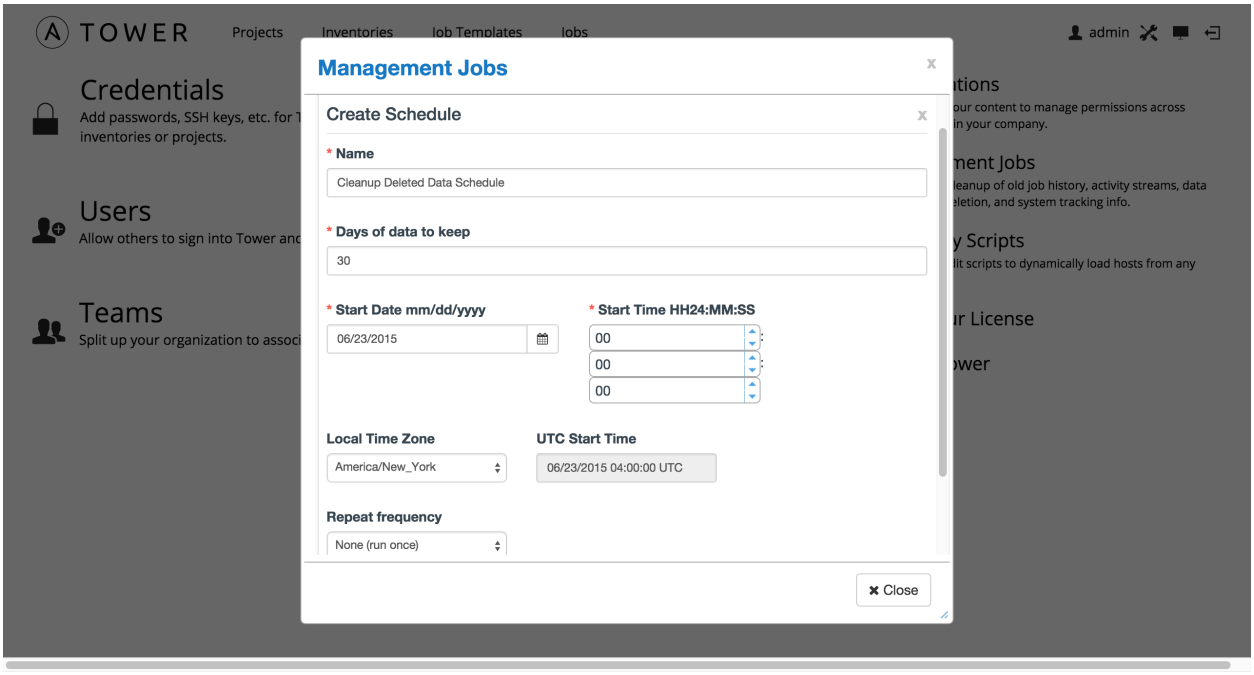
3.2 Removing Data Scheduled for Deletion

To remove object history which has been marked for deletion (much like emptying your trashcan in a desktop environment), click on the  button beside **Cleanup Deleted Data**.



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

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



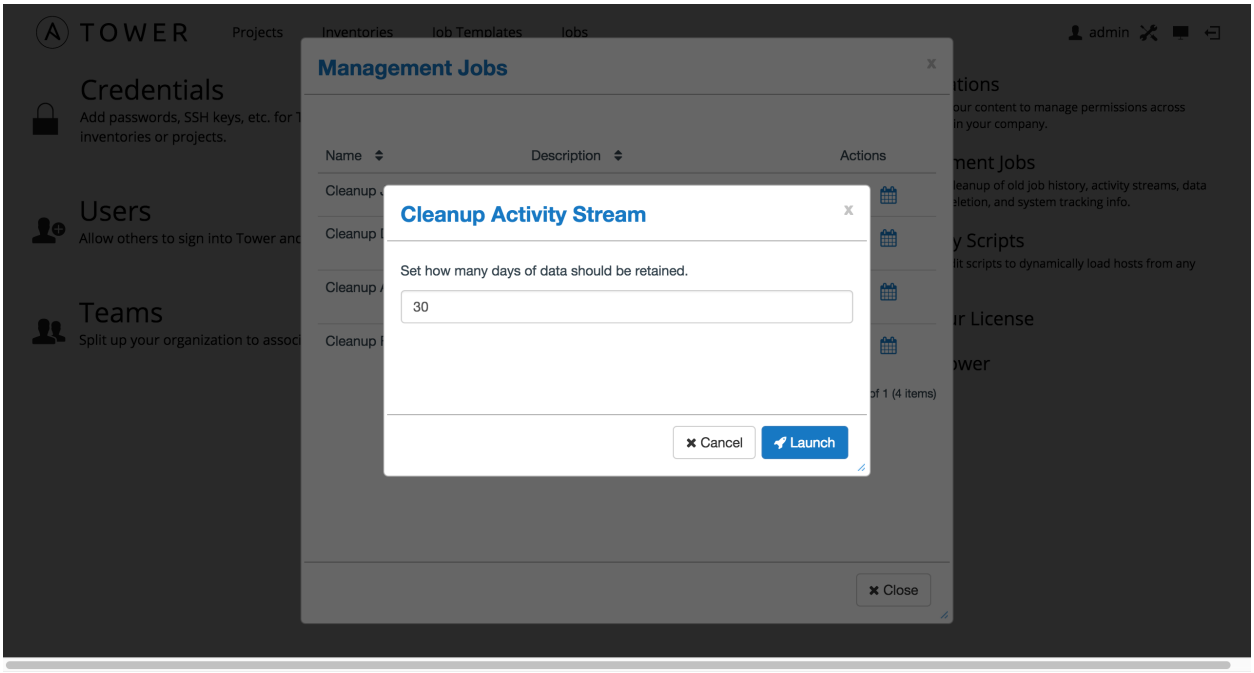
Enter the appropriate details for:

- **Name**: populated by default to match the management job type and can be left alone
- **Days of data to keep**: Enter the number of days of data you would like to retain.
- **Start Date**: Enter the date for which you want data cleanup to start (month/day/year format)
- **Start Time**: Enter the time to start data cleanup in hours, minutes, and seconds.
- **Local Time Zone**: Select your preferred time zone.
- **UTC Start Time**: Review the UTC Start time which is based on the Start Time and Local Time Zone you select.
- **Repeat Frequency**: Select how often you want the cleanup job to run. You can enter cleanups to occur once or every *X* minutes, hours, days, weeks, months, or years.

The **View Details** link at the bottom displays a description of the schedule and a list of the scheduled occurrences in the selected Local Time Zone.

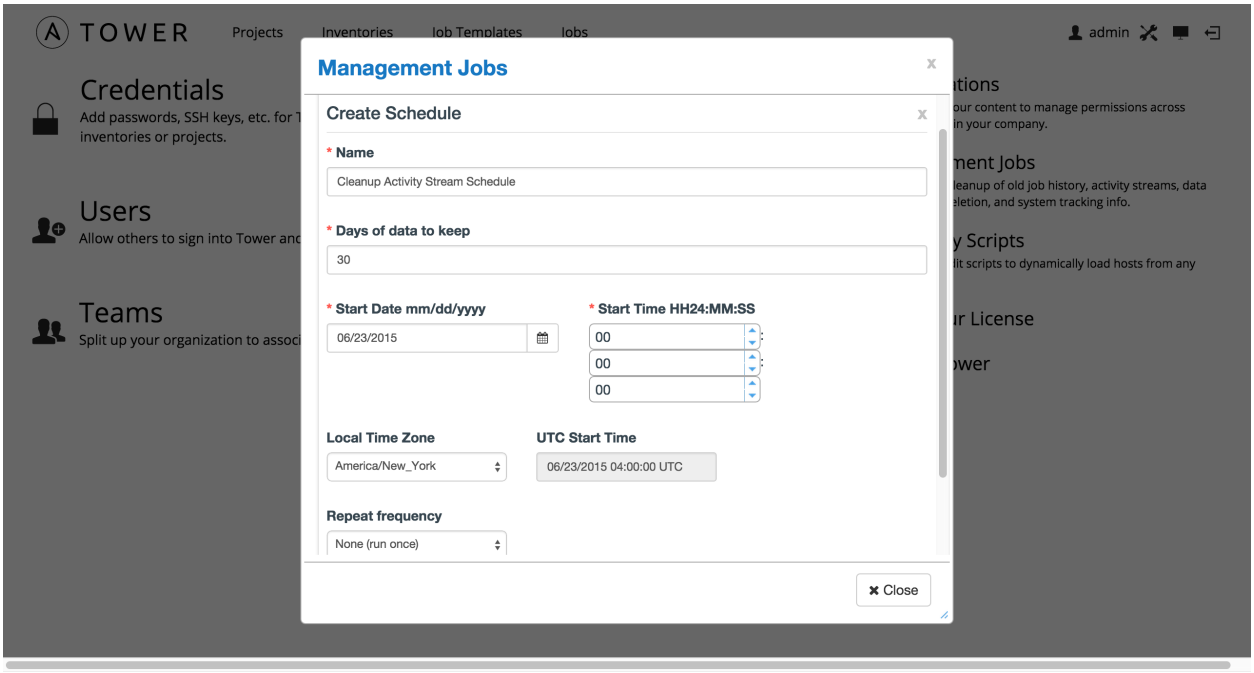
3.3 Removing Old Activity Stream Data

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**.

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



Enter the appropriate details for:

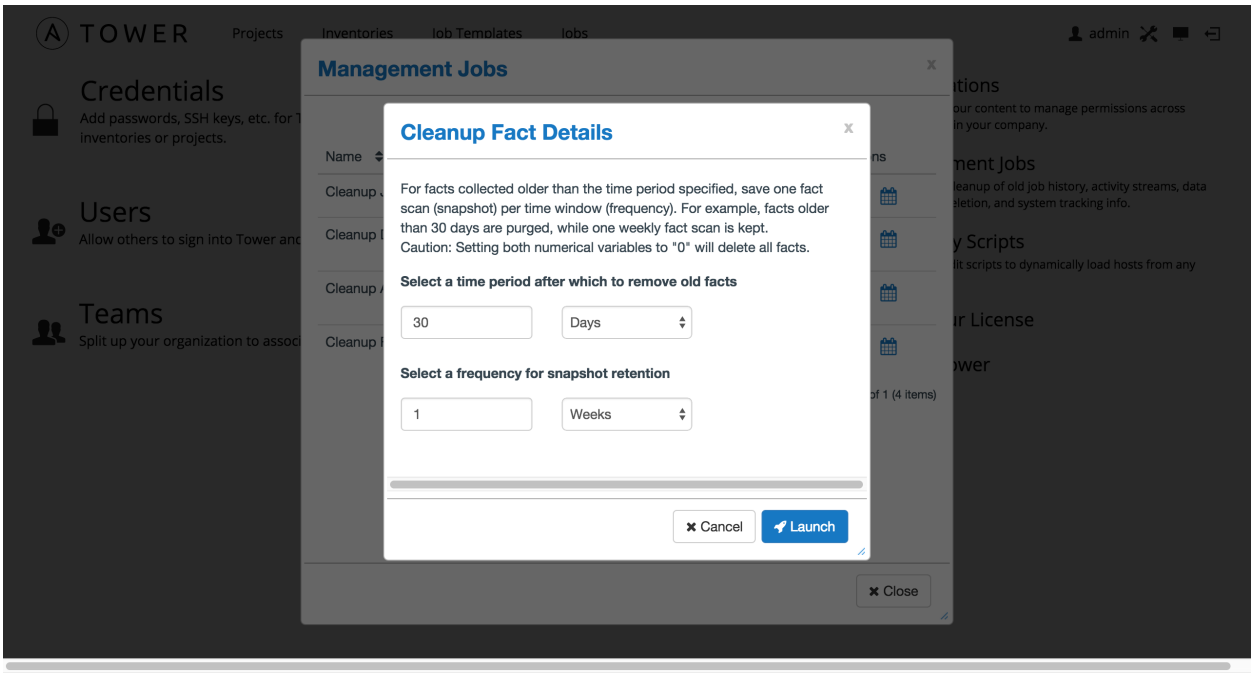
- **Name**: populated by default to match the management job type and can be left alone
- **Days of data to keep**: Enter the number of days of data you would like to retain.
- **Start Date**: Enter the date for which you want data cleanup to start (month/day/year format)
- **Start Time**: Enter the time to start data cleanup in hours, minutes, and seconds.

- **Local Time Zone**: Select your preferred time zone.
- **UTC Start Time**: Review the UTC Start time which is based on the Start Time and Local Time Zone you select.
- **Repeat Frequency**: Select how often you want the cleanup job to run. You can enter cleanups to occur once or every *X* minutes, hours, days, weeks, months, or years.

The **View Details** link at the bottom displays a description of the schedule and a list of the scheduled occurrences in the selected Local Time Zone.

3.4 Removing Old System Tracking Data

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

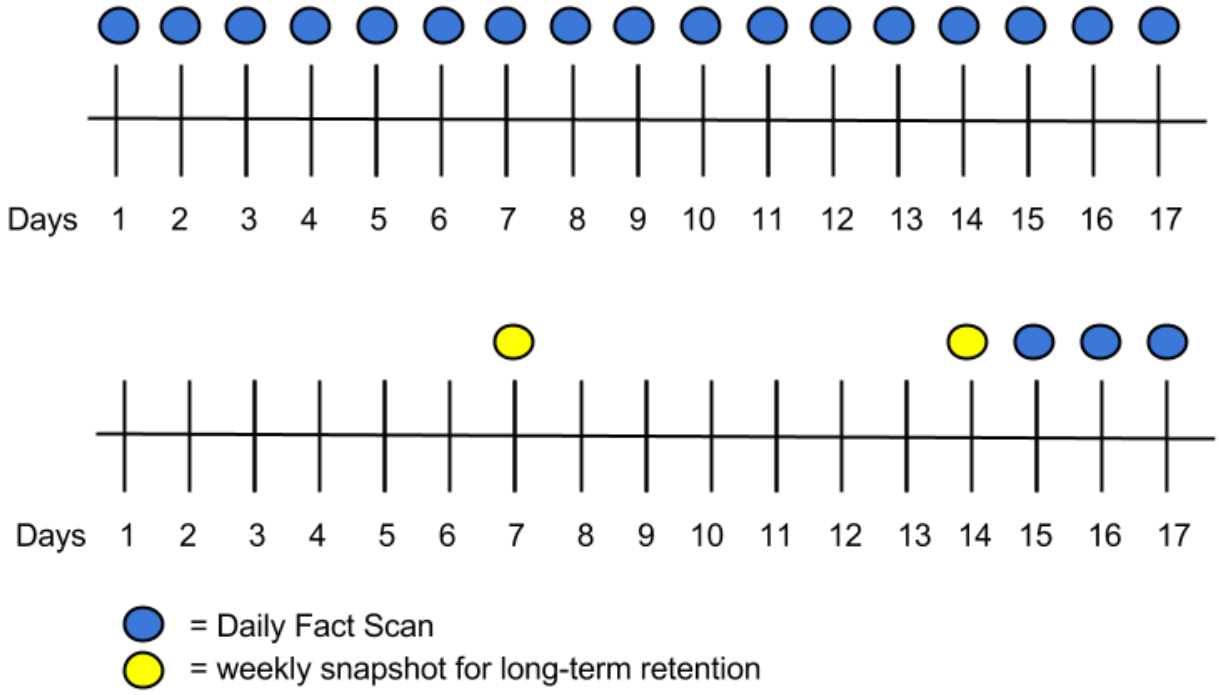


Select the **time period** after which you want to remove old data as well as the **frequency** for snapshot retention.

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.

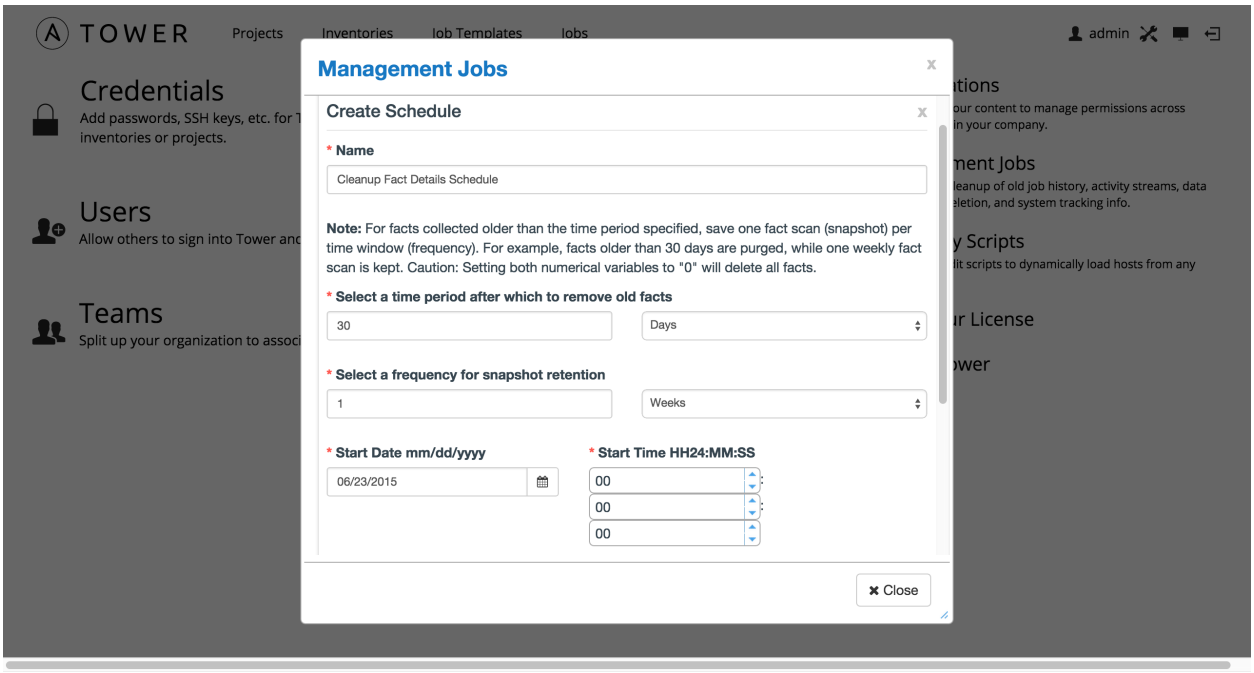
> **Warning:** Setting both numerical variables to “0” will delete all facts.

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.

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



Enter the appropriate details for:

- **Name**: populated by default to match the management job type and can be left alone
- **Select a time period after which to remove old facts**: Select the number of days/weeks/years, after which old data will be removed.
- **Select a frequency for snapshot retention**: Select how often (days/weeks/years) to keep snapshots of your data.
- **Start Date**: Enter the date for which you want data cleanup to start (month/day/year format)
- **Start Time**: Enter the time to start data cleanup in hours, minutes, and seconds.
- **Local Time Zone**: Select your preferred time zone.
- **UTC Start Time**: Review the UTC Start time which is based on the Start Time and Local Time Zone you select.
- **Repeat Frequency**: Select how often you want the cleanup job to run. You can enter cleanups to occur once or every *X* minutes, hours, days, weeks, months, or years.

The **View Details** link at the bottom displays a description of the schedule and a list of the scheduled occurrences in the selected Local Time Zone.

CHAPTER

FOUR

USING LDAP WITH TOWER

Note: LDAP support is only available to those with Enterprise-level licenses.

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, if you need help, contact Ansible via the Red Hat Customer Portal at https://access.redhat.com/.

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

CHAPTER

FIVE

HIGH AVAILABILITY

Tower installations can occur in a High Availability (HA) configuration. In this configuration, Tower runs with a single active node, called the Primary instance, and any number of inactive nodes, called Secondary instances. Secondary instances can become Primary at any time, with certain caveats. Running in a high-availability setup requires any database that Tower uses to be external–Postgres and MongoDB must be installed on a machine that is not one of the primary or secondary tower nodes.

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

For instructions on how to install into a HA configuration, refer to the Ansible Tower Installation and Reference Guide.

Note: High Availability support is only available to those with Enterprise-level licenses.

5.1 Setup Considerations

When creating a HA 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, it does not support replacing MongoDB with a different component.

- Tower servers need isolation.

  If the primary and secondary Tower services share a physical host, a network, or potentially a datacenter, 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 an HA mode, but the database is not run in an HA 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. 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 HA 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 HA mode, the remote Postgres and MongoDB version requirements are *Postgresql 9.4.x* and *mongodb 3.0.x*.

  Postgresql 9.4.x and Mongodb 3.0.x are 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, an HA 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 a MongoDB setup, refer to Requirements in the *Ansible Tower Installation and Reference Guide*.

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 HA configuration of Tower

To see the HA 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 HA 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.

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.

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']`.

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

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.

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

TOWER-MANAGE

tower-manage (formerly awx-manage) is a utility 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.

`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`).

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.

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 *High Availability* 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

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.

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
```

9.1 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"
    ;;
```

Note: The backup playbook does not back up the `/var/lib/awx/projects/` directory. If you have any projects of the type `manual` (projects that do not use source control), you must have a separate backup and restore procedure for them.

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 ensures that the backup file exists before beginning the restoration. It 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.

9.2 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

TEN

TROUBLESHOOTING, TIPS, AND TRICKS

10.1 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/.

10.2 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.)

10.3 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.

10.4 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`.

10.5 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.

10.6 Locate and configure the 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.

10.7 Playbook Stays in Pending

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

- Run `ansible-tower-service restart` on the Tower server.
- Check to ensure that the `/var/` partition has more than 1GB of space available. Jobs will not complete with insufficient space on the `/var/` partition.
- Ensure all supervisor services are running via `supervisorctl status`.
- Check to see if disabling PRoot solves your issues (and, if so, please report back to Ansible's Support team to let them know it helped). Navigate to the `/etc/tower/settings.py` file and set `AWX_PROOT_ENABLED=False`, then restart services with the `ansible-tower-service-restart` command.

If you continue to have problems, run `sosreport` as root on the Tower server, then file a support request with the result (contact Ansible via the Red Hat Customer Portal at https://access.redhat.com/).

10.8 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 does cause Ansible to exit, Ansible is designed to finish tasks before it exits and only does so *after* the currently running play has completed.

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

10.9 Change the default timeout for authentication

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

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

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

Warning: When including a `local_config.js` file with specifically configured variables, it will overwrite Tower's default `config.js` file. You should copy the default `config.js` file over to `local_config.js` before making changes for specific variables, ensuring that everything Tower needs to call from that file is available.

Tower is designed to look for the `local_config.js` file first. If this file is not found, it uses the default `config.js` file included with Tower. If `local_config.js` is found, it is used *instead* of `config.js`.

The variables you can edit within the `local_config.js` 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: -!$%^&*()_+|~=`{}[]:";'<>?,./
- `session_timeout: 1800` – Number of seconds before an inactive session is automatically timed out and forced to log in again. This is separate from time out value set in API.

10.10 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/
```

10.11 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.

CHAPTER

ELEVEN

INDEX

- genindex

Third Party Rights

INDEX

M

O

P

R

S

T