Ansible Tower Administration Guide

Release Ansible Tower 3.8.4

Red Hat, Inc.

Apr 18, 2023

CONTENTS

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

The *Ansible Tower Administration Guide* documents the administration of Ansible Tower through custom scripts, management jobs, and more. Written for DevOps engineers and administrators, the *Ansible Tower Administration Guide* assumes a basic understanding of the systems requiring management with Tower's easy-to-use graphical interface. This document has been updated to include information for the latest release of Ansible Tower v3.8.4.

We Need Feedback!

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

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

Ansible Tower Version 3.8.4; September 08, 2021; https://access.redhat.com/

CHAPTER

ONE

TOWER LICENSING, UPDATES, AND SUPPORT

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

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

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

1.1 Support

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

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

1.2 Trial / Evaluation

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

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

1.3 Subscription Types

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

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

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

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

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

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

1.4 Node Counting in Licenses

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

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

1.5 Attaching Subscriptions

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

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

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

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

The command returns the following:

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

(continues on next page)

(continued from previous page)

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

To attach this subscription:

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

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

To check whether the subscription was successfully attached:

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

To remove this subscription:

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

1.6 Tower Component Licenses

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

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

CHAPTER

TWO

STARTING, STOPPING, AND RESTARTING TOWER

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

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

Note: In clustered installs, `ansible-tower-service restart` does not include PostgreSQL as part of the services that are restarted because it exists external to Tower, and because PostgreSQL does not always require a restart. Use `systemctl restart ansible-tower` to restart services on clustered environments instead. Also you must restart each cluster node for certain changes to persist as opposed to a single node for a localhost install. For more information on clustered environments, see the *Clustering* section.

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

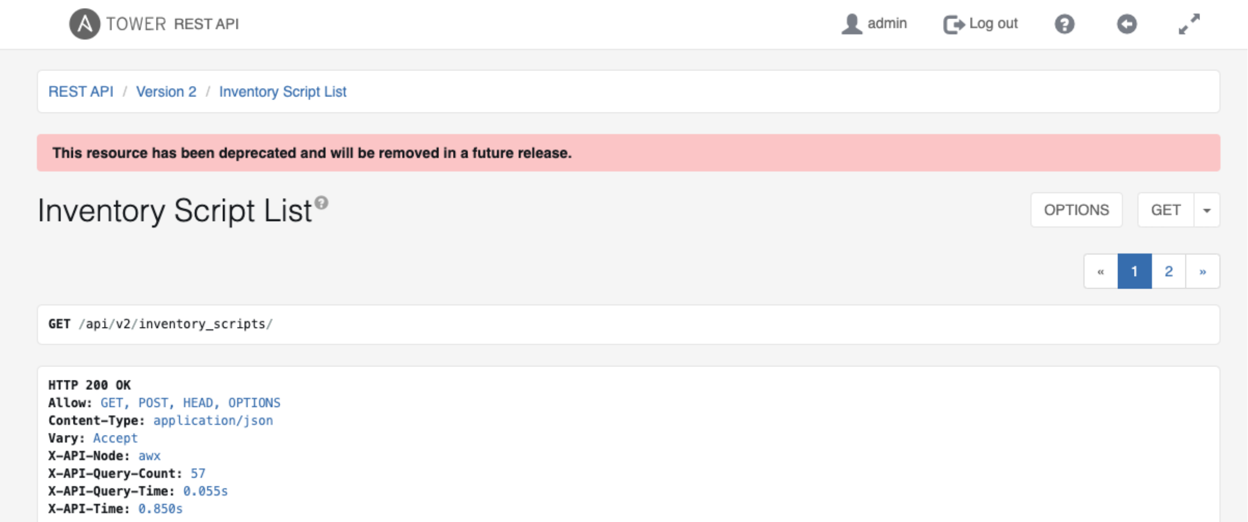
Note: When running Tower containerized in *OpenShift*, do not use the `ansible-tower-service` script. Restart the pod using OpenShift instead.

CHAPTER

THREE

CUSTOM INVENTORY SCRIPTS

Note: Inventory scripts are deprecated as of Ansible Tower Version 3.8.4.



This resource has been deprecated and will be removed in a future release.

Inventory Script List

```
GET /api/v2/inventory_scripts/
```

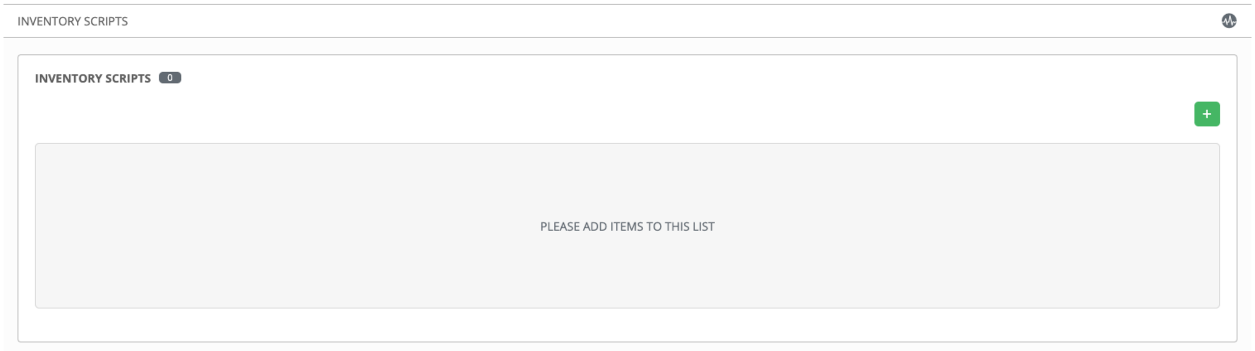
```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
X-API-Node: awx
X-API-Query-Count: 57
X-API-Query-Time: 0.055s
X-API-Time: 0.850s
```

If you use custom inventory scripts, migrate to sourcing these scripts from a project. See *Inventory File Importing* in the subsequent section, and also refer to Inventory Sources in the *Ansible Tower User Guide* for more detail.

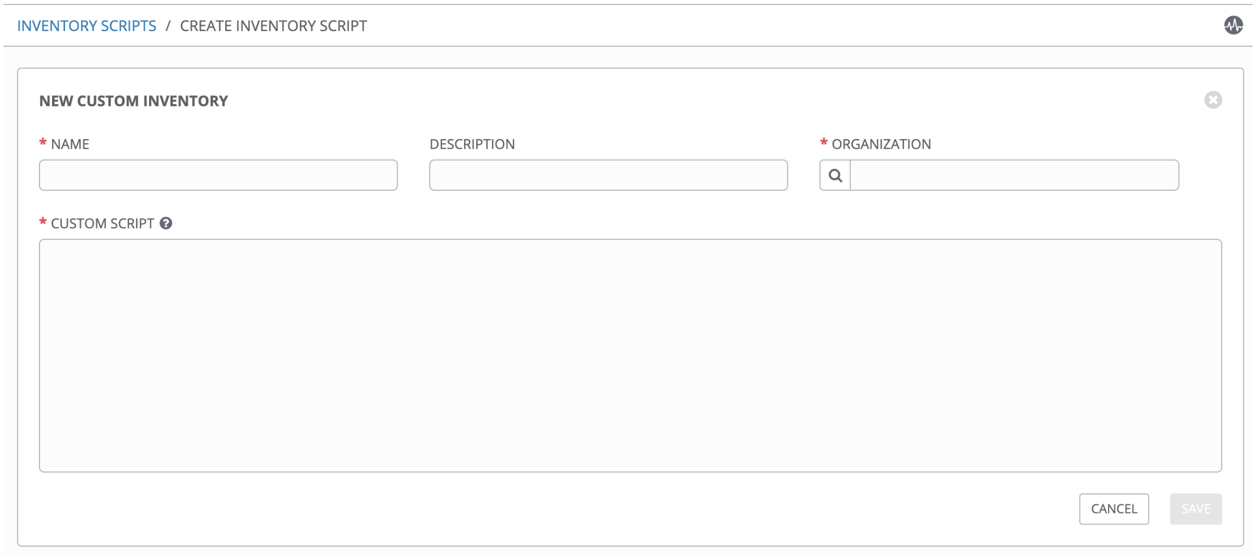
Tower includes built-in support for syncing dynamic inventory from cloud sources such as Amazon AWS, Google Compute Engine, among others. Tower also offers the ability to use a custom script to pull from your own inventory source.

Note: Edits and additions to Inventory host variables persist beyond an inventory sync as long as `--overwrite_vars` is **not** set.

To manage the custom inventory scripts available in Tower, click the Inventory Scripts (</>) icon from the left navigation bar.

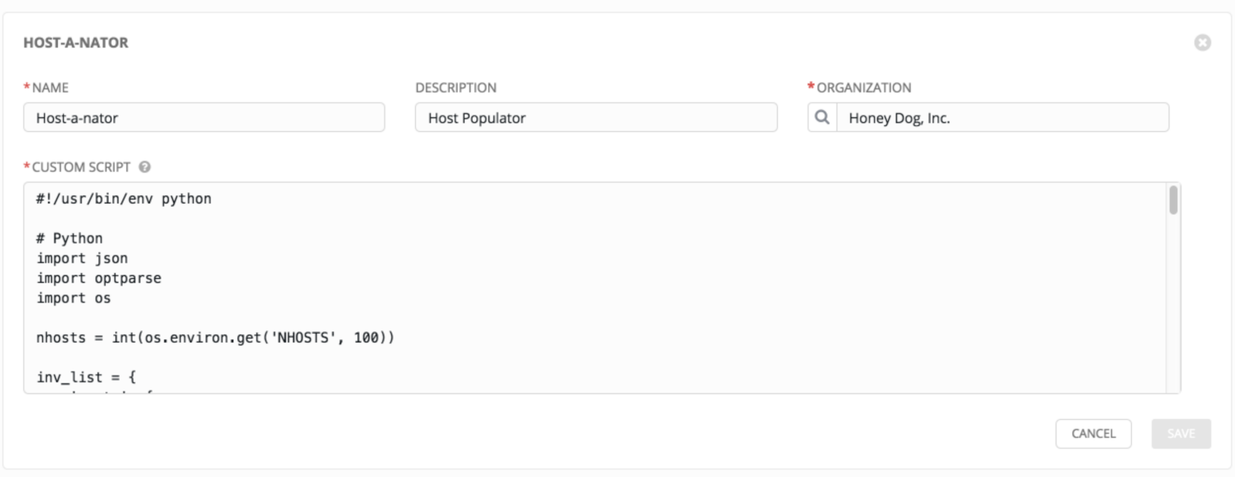


To add a new custom inventory script, click the + button.



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.



*NAME Host-a-nator

DESCRIPTION Host Populator

*ORGANIZATION Honey Dog, Inc.

*CUSTOM SCRIPT

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

# Python
import json
import optparse
import os

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

inv_list = {
```

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/conf.d/custom.py`.

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

CHAPTER

FOUR

INVENTORY FILE IMPORTING

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

4.1 Custom Dynamic Inventory Scripts

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

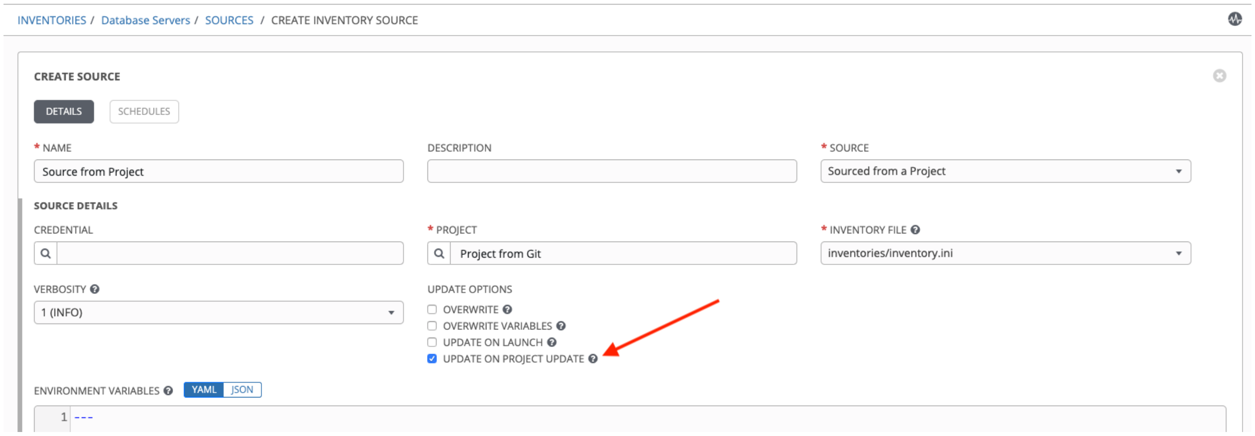
You can specify environment variables when you edit the SCM inventory source itself. For some scripts, this will be sufficient, however, this is not a secure way to store secret information that gives access to cloud providers or inventory.

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

For more detail, refer to Credential types.

4.1.1 Update on Project Update

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



INVENTORIES / Database Servers / SOURCES / CREATE INVENTORY SOURCE

CREATE SOURCE

DETAILS | SCHEDULES

* NAME: Source from Project

DESCRIPTION:

* SOURCE: Sourced from a Project

SOURCE DETAILS

CREDENTIAL:

* PROJECT: Project from Git

* INVENTORY FILE: inventories/inventory.ini

VERBOSITY: 1 (INFO)

UPDATE OPTIONS

□ OVERWRITE
□ OVERWRITE VARIABLES
□ UPDATE ON LAUNCH
☑ UPDATE ON PROJECT UPDATE

ENVIRONMENT VARIABLES: YAML | JSON

```
---
```

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

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

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

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

Also, projects are not blocked from updating while a related job is running. In cases where you have a big project (around 10 GB), disk space on `/tmp` may be an issue.

4.2 SCM Inventory Source Fields

The source fields used are:

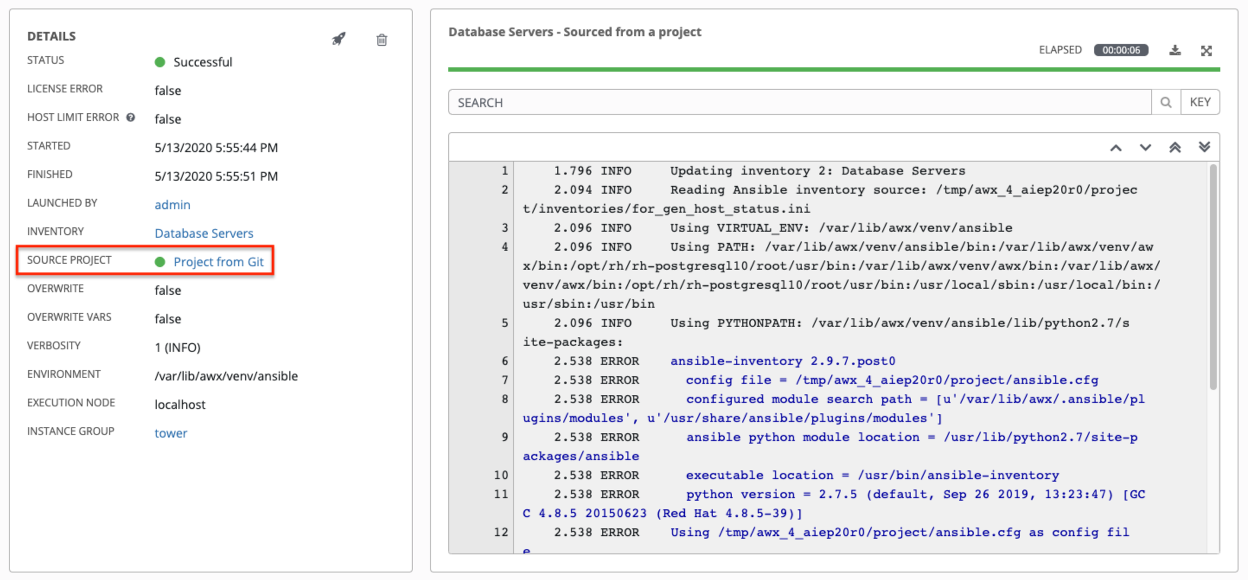
- `source_project`: project to use
- `source_path`: relative path inside the project indicating a directory or a file. If left blank, "" is still a relative path indicating the root directory of the project
- `source_vars`: if set on a "file" type inventory source then they will be passed to the environment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

An update of the project automatically triggers an inventory update where it is used. An update of the project is scheduled immediately after creation of the inventory source. Neither inventory nor project updates are blocked while a related job is running. In cases where you have a big project (around 10 GB), disk space on `/tmp` may be an issue.

You can specify a location manually in the Tower User Interface from the Create Inventory Source page. Refer to the Inventories section of the *Ansible Tower User Guide* for instructions on creating an inventory source.

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

For inventories with SCM sources, the Job Details page for inventory updates show a status indicator for the project update as well as the name of the project. The status indicator links to the project update job. The project name links to the project.



DETAILS

STATUS	● Successful
LICENSE ERROR	false
HOST LIMIT ERROR	false
STARTED	5/13/2020 5:55:44 PM
FINISHED	5/13/2020 5:55:51 PM
LAUNCHED BY	admin
INVENTORY	Database Servers
SOURCE PROJECT	● Project from Git
OVERWRITE	false
OVERWRITE VARS	false
VERBOSITY	1 (INFO)
ENVIRONMENT	/var/lib/awx/venv/ansible
EXECUTION NODE	localhost
INSTANCE GROUP	tower

Database Servers - Sourced from a project

ELAPSED 00:00:06

```
    1.796 INFO     Updating inventory 2: Database Servers
    2.094 INFO     Reading Ansible inventory source: /tmp/awx_4_aiep20r0/projec
t/inventories/for_gen_host_status.ini
    2.096 INFO     Using VIRTUAL_ENV: /var/lib/awx/venv/ansible
    2.096 INFO     Using PATH: /var/lib/awx/venv/ansible/bin:/var/lib/awx/venv/aw
x/bin:/opt/rh/rh-postgresql10/root/usr/bin:/var/lib/awx/venv/awx/bin:/var/lib/awx/
venv/awx/bin:/opt/rh/rh-postgresql10/root/usr/bin:/usr/local/sbin:/usr/local/bin:/
usr/sbin:/usr/bin
    2.096 INFO     Using PYTHONPATH: /var/lib/awx/venv/ansible/lib/python2.7/s
ite-packages:
    2.538 ERROR    ansible-inventory 2.9.7.post0
    2.538 ERROR      config file = /tmp/awx_4_aiep20r0/project/ansible.cfg
    2.538 ERROR      configured module search path = [u'/var/lib/awx/.ansible/pl
ugins/modules', u'/usr/share/ansible/plugins/modules']
    2.538 ERROR      ansible python module location = /usr/lib/python2.7/site-p
ackages/ansible
    2.538 ERROR      executable location = /usr/bin/ansible-inventory
    2.538 ERROR      python version = 2.7.5 (default, Sep 26 2019, 13:23:47) [GC
C 4.8.5 20150623 (Red Hat 4.8.5-39)]
    2.538 ERROR    Using /tmp/awx_4_aiep20r0/project/ansible.cfg as config fil
e
```

An inventory update can be performed while a related job is running.

4.2.1 Supported File Syntax

Ansible Tower uses the `ansible-inventory` module from Ansible to process inventory files, and supports all valid inventory syntax that Tower requires.

CHAPTER

FIVE

MULTI-CREDENTIAL ASSIGNMENT

Ansible Tower provides support for assigning zero or more credentials to a job template.

5.1 Background

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

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

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

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

5.2 Important Changes

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

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

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

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

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

5.3 Launch Time Considerations

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

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

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

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

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

5.4 Multi-Vault Credentials

As it possible to assign multiple credentials to a job, you can specify multiple Vault credentials to decrypt when your job template runs. This functionality mirrors the support for multiple vault passwords for a playbook run in Ansible 2.4 and later.

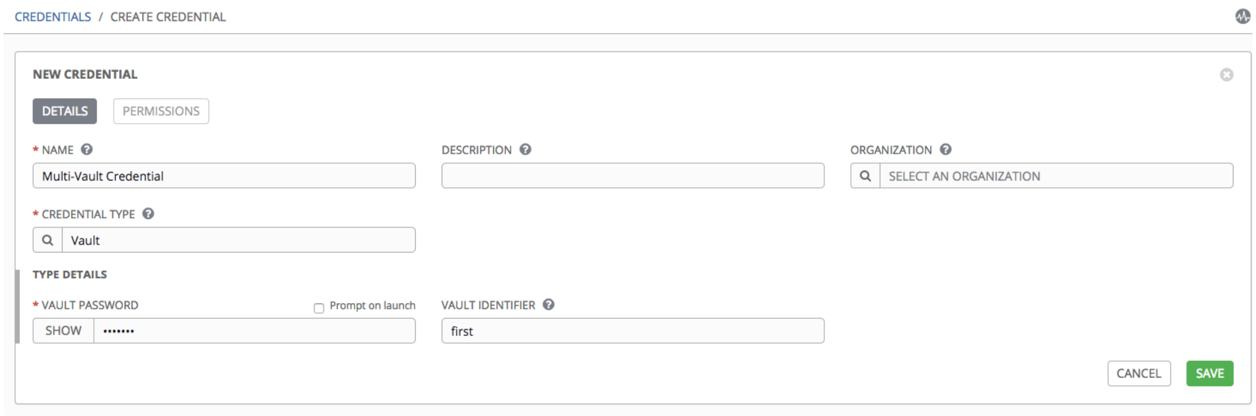
Vault credentials now have an optional field, `vault_id`, which is analogous to the `--vault-id` argument to `ansible-playbook`. To run a playbook which makes use of multiple vault passwords:

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

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

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

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



* NAME: Multi-Vault Credential

DESCRIPTION:

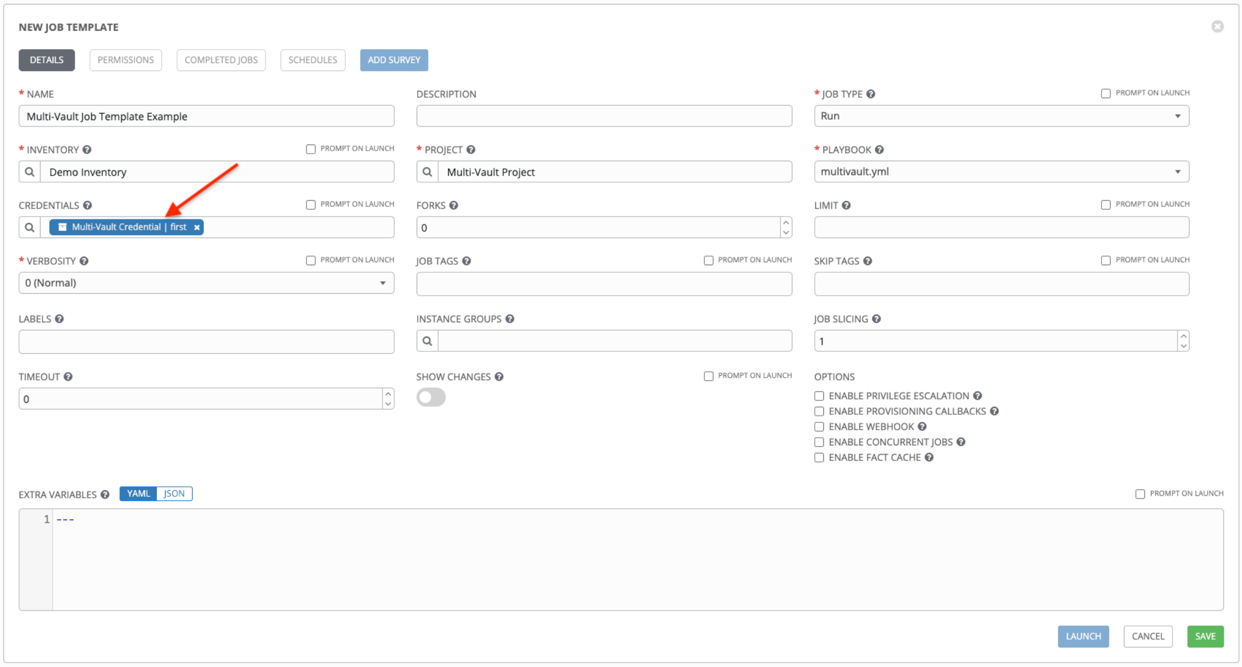
ORGANIZATION: SELECT AN ORGANIZATION

* CREDENTIAL TYPE: Vault

* VAULT PASSWORD: ••••••• □ Prompt on launch

VAULT IDENTIFIER: first

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

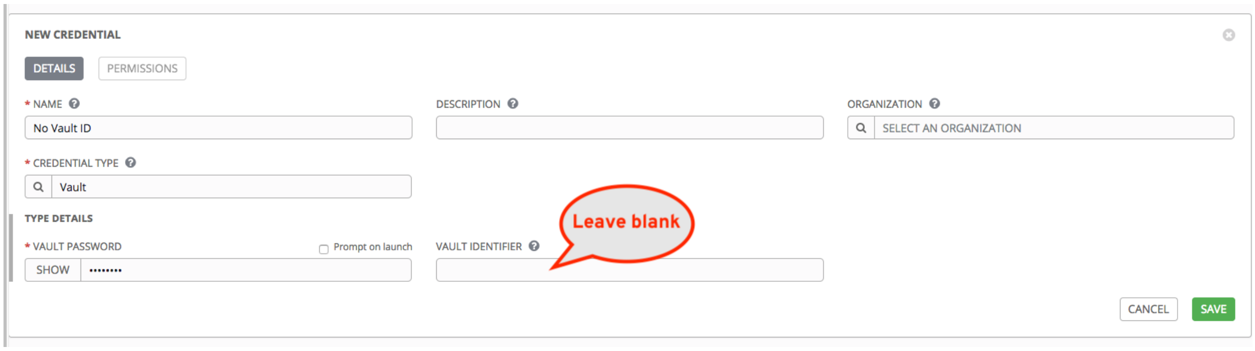


NEW JOB TEMPLATE

- * NAME: Multi-Vault Job Template Example
- DESCRIPTION:
- * JOB TYPE: Run
- * INVENTORY: Demo Inventory
- * PROJECT: Multi-Vault Project
- * PLAYBOOK: multivault.yml
- CREDENTIALS: Multi-Vault Credential | first
- FORKS: 0
- LIMIT:
- * VERBOSITY: 0 (Normal)
- JOB TAGS:
- SKIP TAGS:
- LABELS:
- INSTANCE GROUPS:
- JOB SLICING: 1
- TIMEOUT: 0
- SHOW CHANGES:
- OPTIONS:
  - □ ENABLE PRIVILEGE ESCALATION
  - □ ENABLE PROVISIONING CALLBACKS
  - □ ENABLE WEBHOOK
  - □ ENABLE CONCURRENT JOBS
  - □ ENABLE FACT CACHE
- EXTRA VARIABLES (YAML):

```
1 ---
```

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



NEW CREDENTIAL

- * NAME: No Vault ID
- DESCRIPTION:
- ORGANIZATION: SELECT AN ORGANIZATION
- * CREDENTIAL TYPE: Vault

TYPE DETAILS

- * VAULT PASSWORD: ••••••••
- VAULT IDENTIFIER:

5.4.1 Prompted Vault Credentials

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

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

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

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

5.4.2 Linked credentials

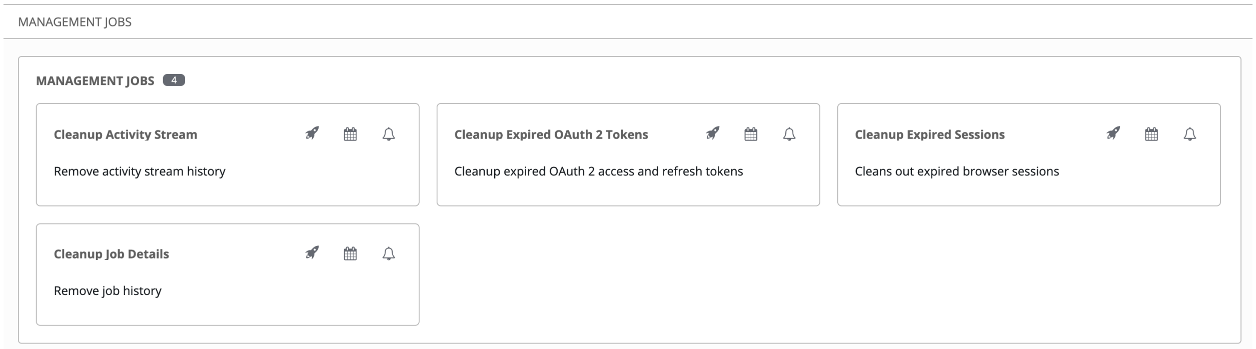
Instead of uploading sensitive credential information into Tower, you can link credential fields to external systems and using them to run your playbooks. Refer to the Secret Management System section of the *Ansible Tower User Guide*.

CHAPTER

SIX

MANAGEMENT JOBS

Management Jobs assist in the cleaning of old data from Tower, including system tracking information, tokens, 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. Click the Management Jobs ( ) icon from the left navigation bar.



MANAGEMENT JOBS

MANAGEMENT JOBS 4

Cleanup Activity Stream
Remove activity stream history

Cleanup Expired OAuth 2 Tokens
Cleanup expired OAuth 2 access and refresh tokens

Cleanup Expired Sessions
Cleans out expired browser sessions

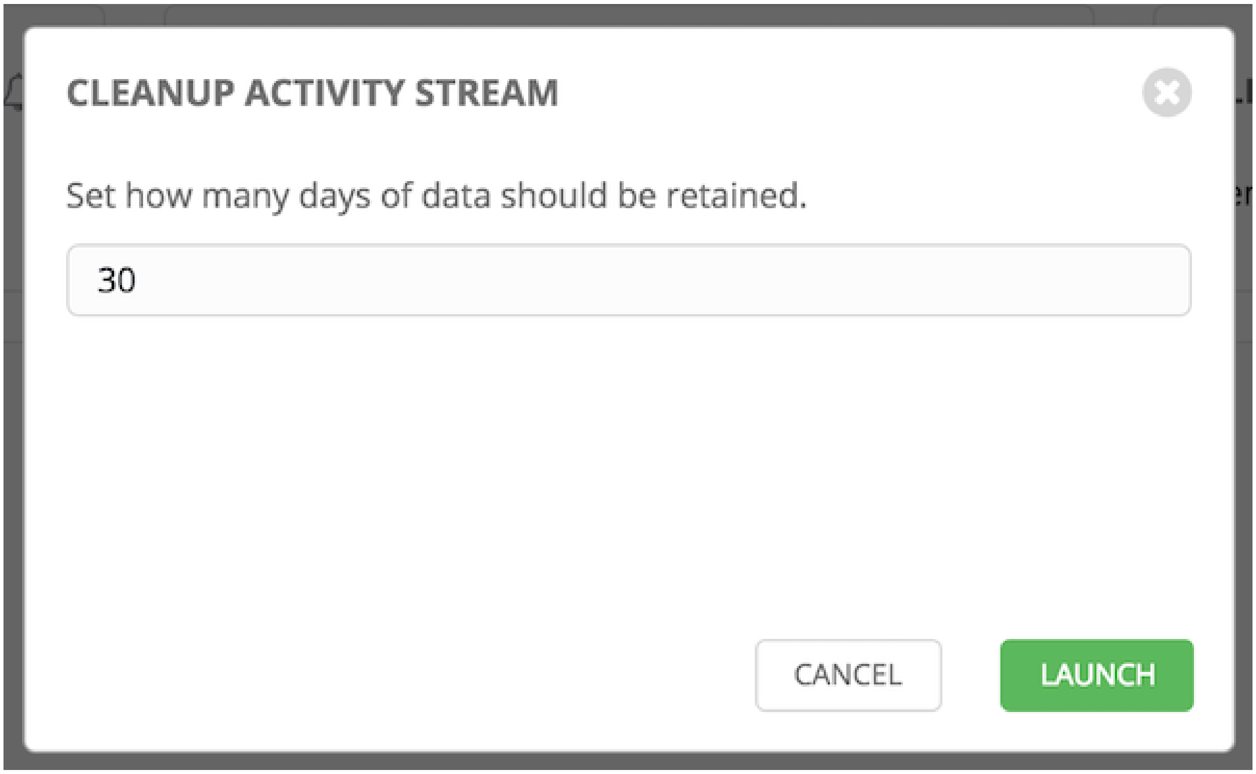
Cleanup Job Details
Remove job history

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

- **Cleanup Activity Stream**: Remove activity stream history older than a specified number of days
- **Cleanup Expired OAuth 2 Tokens**: Remove expired OAuth 2 access tokens and refresh tokens
- **Cleanup Expired Sessions**: Remove expired browser sessions from the database
- **Cleanup Job Details**: Remove job history older than a specified number of days

6.1 Removing Old Activity Stream Data

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



CLEANUP ACTIVITY STREAM

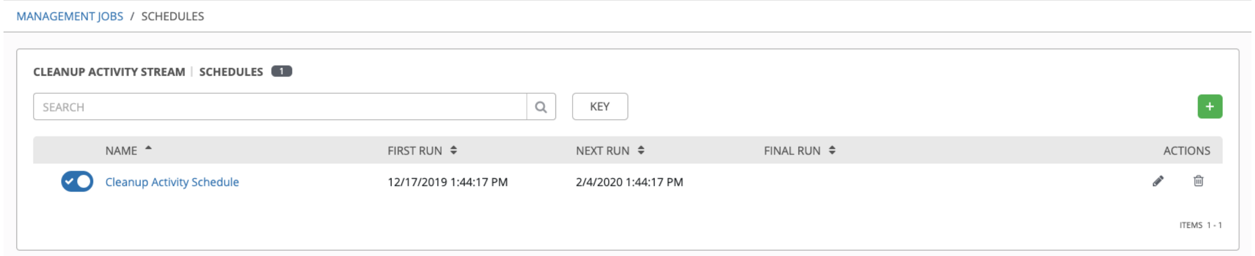
Set how many days of data should be retained.

30

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

6.1.1 Scheduling

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



MANAGEMENT JOBS / SCHEDULES

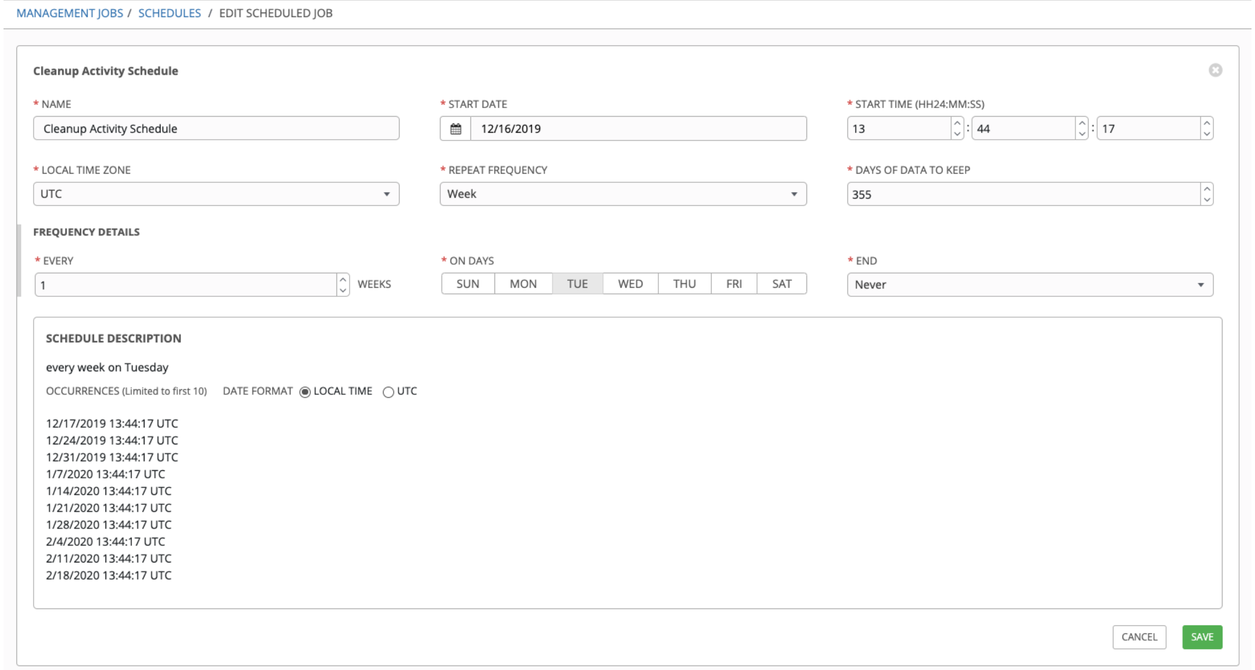
CLEANUP ACTIVITY STREAM | SCHEDULES 1

	NAME	FIRST RUN	NEXT RUN	FINAL RUN	ACTIONS
✓	Cleanup Activity Schedule	12/17/2019 1:44:17 PM	2/4/2020 1:44:17 PM		

ITEMS 1 - 1

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

Click on the Job Name, in this example "Cleanup Activity Schedule", to review or edit the schedule settings. You can also use the + button to create a new schedule for this management job.



MANAGEMENT JOBS / SCHEDULES / EDIT SCHEDULED JOB

Cleanup Activity Schedule

* NAME: Cleanup Activity Schedule

* START DATE: 12/16/2019

* START TIME (HH24:MM:SS): 13 : 44 : 17

* LOCAL TIME ZONE: UTC

* REPEAT FREQUENCY: Week

* DAYS OF DATA TO KEEP: 355

FREQUENCY DETAILS

* EVERY: 1 WEEKS

* ON DAYS: SUN MON [✓ TUE] WED THU FRI SAT

* END: Never

SCHEDULE DESCRIPTION

every week on Tuesday

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

12/17/2019 13:44:17 UTC
12/24/2019 13:44:17 UTC
12/31/2019 13:44:17 UTC
1/7/2020 13:44:17 UTC
1/14/2020 13:44:17 UTC
1/21/2020 13:44:17 UTC
1/28/2020 13:44:17 UTC
2/4/2020 13:44:17 UTC
2/11/2020 13:44:17 UTC
2/18/2020 13:44:17 UTC

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

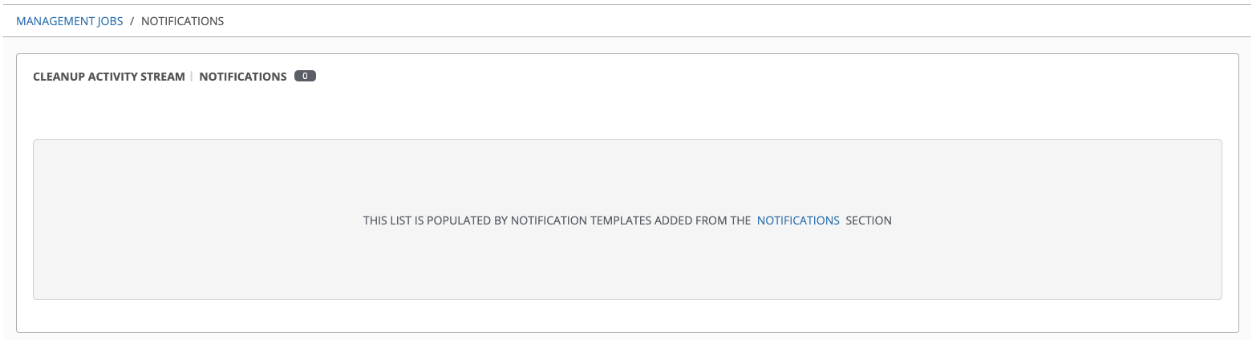
- Name (required)
- Start Date (required)
- Start Time (required)
- Local Time Zone (the entered Start Time should be in this timezone)
- Repeat Frequency (the appropriate options display as the update frequency is modified.)

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

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

6.1.2 Notifications

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



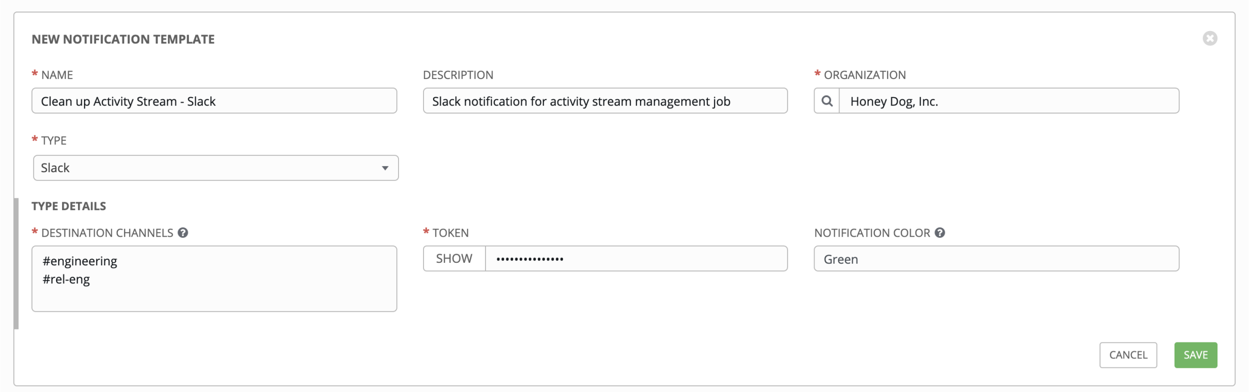
MANAGEMENT JOBS / NOTIFICATIONS

CLEANUP ACTIVITY STREAM | NOTIFICATIONS 0

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

If none exist, click the **Notifications** link to create a new notification. Notification types include:

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



NEW NOTIFICATION TEMPLATE

* NAME: Clean up Activity Stream - Slack

DESCRIPTION: Slack notification for activity stream management job

* ORGANIZATION: Honey Dog, Inc.

* TYPE: Slack

TYPE DETAILS

* DESTINATION CHANNELS: #engineering
#rel-eng

* TOKEN: SHOW ••••••••••••••••

NOTIFICATION COLOR: Green

CANCEL SAVE

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

6.2 Cleanup Expired OAuth2 Tokens

To remove expired OAuth2 tokens, click on the launch (🚀) button beside **Cleanup Expired OAuth2 Tokens**.

You can review or set a schedule for cleaning up expired OAuth2 tokens by performing the same procedure described for activity stream management jobs. See *Scheduling* for detail.

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

6.3 Cleanup Expired Sessions

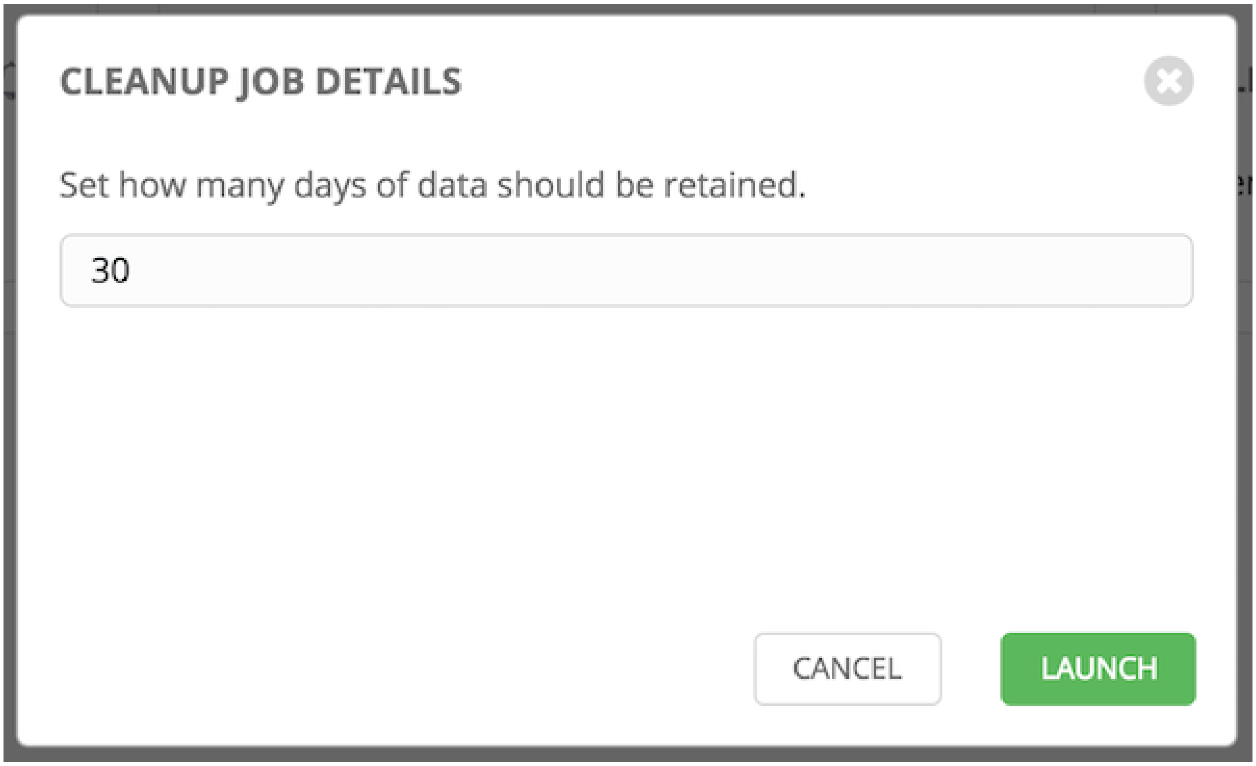
To remove expired sessions, click on the launch (🚀) button beside **Cleanup Expired Sessions**.

You can review or set a schedule for cleaning up expired sessions by performing the same procedure described for activity stream management jobs. See *Scheduling* for detail.

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

6.4 Removing Old Job History

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



CLEANUP JOB DETAILS

Set how many days of data should be retained.

30

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

Note: The initial job run for a Tower resource (e.g. Projects, Job Templates) is excluded from **Cleanup Job Details**, regardless of retention value.

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

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

CHAPTER

SEVEN

CLUSTERING

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

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

Each instance should be able to join the Tower cluster and expand its ability to execute jobs. This is a simple system where jobs can and will run anywhere rather than be directed on where to run. Also, clustered instances can be grouped into different pools/queues, called *Instance Groups*.

Tower supports container-based clusters using OpenShift, meaning new Tower instances can be installed on this platform without any variation or diversion in functionality. However, there are certain *OpenShift Deployment and Configuration* differences that must be considered. You can create instance groups to point to a Kubernetes container. For more detail, see the *Execution Environments* section.

Supported Operating Systems

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

- Red Hat Enterprise Linux 7 or later (RHEL8 recommended, can be either RHEL 7 or Centos 7 instances)

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

7.1 Setup Considerations

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

Important considerations to note in the new clustering environment:

- PostgreSQL is still a standalone instance and is not clustered. Tower does not manage replica configuration or database failover (if the user configures standby replicas).
- When spinning up a cluster, the database node should be a standalone server, and PostgreSQL should not be installed on one of the Tower nodes.
- The maximum supported instances in a cluster is 20.
- All instances should be reachable from all other instances and they should be able to reach the database. It is also important for the hosts to have a stable address and/or hostname (depending on how the Tower host is configured).

- All instances must be geographically collocated, with reliable low-latency connections between instances.
- For purposes of upgrading to a clustered environment, your primary instance must be part of the `tower` group in the inventory *AND* it needs to be the first host listed in the `tower` group.
- Manual projects must be manually synced to all instances by the customer, and updated on all instances at once.
- There is no concept of primary/secondary in the new Tower system. All systems are primary.
- The `inventory` file for Tower deployments should be saved/persisted. If new instances are to be provisioned, the passwords and configuration options, as well as host names, must be made available to the installer.

7.2 Install and Configure

Provisioning new instances involves updating the `inventory` file and re-running the setup playbook. It is important that the `inventory` file contains all passwords and information used when installing the cluster or other instances may be reconfigured. The `inventory` file inventory contains a single inventory group, `tower`.

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

```
[tower]
hostA
hostB
hostC

[instance_group_east]
hostB
hostC

[instance_group_west]
hostC
hostD
```

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

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

```
[tower]
hostA
hostB
hostC

[database]
hostDB
```

When a playbook runs on an individual Tower instance in a cluster, the output of that playbook is broadcast to all of the other nodes as part of Tower's websocket-based streaming output functionality. It is best to handle this data broadcast using internal addressing by specifying a private routable address for each node in your inventory:

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

Note: Prior versions of Ansible Tower used the variable name `rabbitmq_host`. If you are upgrading from a previous version of Tower, and you previously specified `rabbitmq_host` in your inventory, simply rename `rabbitmq_host` to `routable_hostname` before upgrading.

7.2.1 Instances and Ports Used by Tower and Automation Hub

Ports and instances used by Tower and also required by the on-premise Automation Hub node are as follows:

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

7.2.2 Optional SSH Authentication

For users who wish to manage SSH authentication from "controller" nodes to "isolated" nodes via some system outside of Tower (such as externally-managed passwordless SSH keys), this behavior can be disabled by running a deprovisioning command: `awx-manage remove_isolated_key`.

7.3 Status and Monitoring via Browser API

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

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

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

7.4 Instance Services and Failure Behavior

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

- HTTP Services - This includes the Tower application itself as well as external web services.
- Callback Receiver - Receives job events from running Ansible jobs.
- Dispatcher - The worker queue that processes and runs all jobs.
- Redis - This key value store is used as a queue for event data propagated from ansible-playbook to the application.

- Rsyslog - log processing service used to deliver logs to various external logging services.

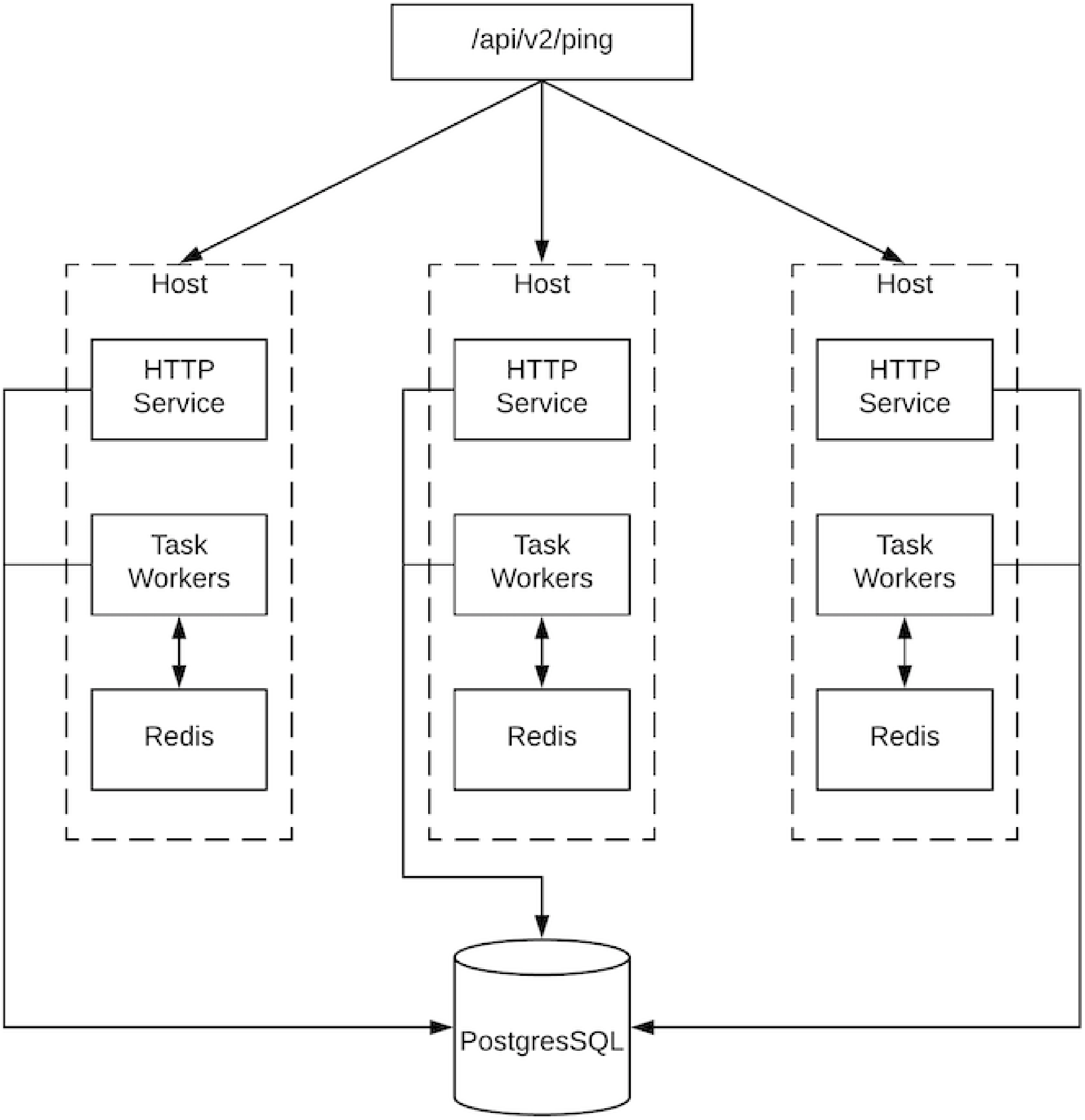
Tower is configured in such a way that if any of these services or their components fail, then all services are restarted. If these fail sufficiently often in a short span of time, then the entire instance will be placed offline in an automated fashion in order to allow remediation without causing unexpected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

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

7.5 Job Runtime Behavior

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

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



- Project updates run successfully on any instance that could potentially run a job. Projects will sync themselves to the correct version on the instance immediately prior to running the job. If the needed revision is already locally checked out and Galaxy or Collections updates are not needed, then a sync may not be performed.
- When the sync happens, it is recorded in the database as a project update with a `launch_type = sync` and `job_type = run`. Project syncs will not change the status or version of the project; instead, they will update the source tree *only* on the instance where they run.
- If updates are needed from Galaxy or Collections, a sync is performed that downloads the required roles, consuming that much more space in your /tmp file. In cases where you have a big project (around 10 GB), disk space on `/tmp` may be an issue.

7.5.1 Job Runs

By default, when a job is submitted to the tower queue, it can be picked up by any of the workers. However, you can control where a particular job runs, such as restricting the instances from which a job runs on.

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

7.6 Deprovision Instances

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

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

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

Similarly, deprovisioning instance groups in Tower does not automatically deprovision or remove instance groups. For more information, refer to the *Deprovision Instance Groups* section.

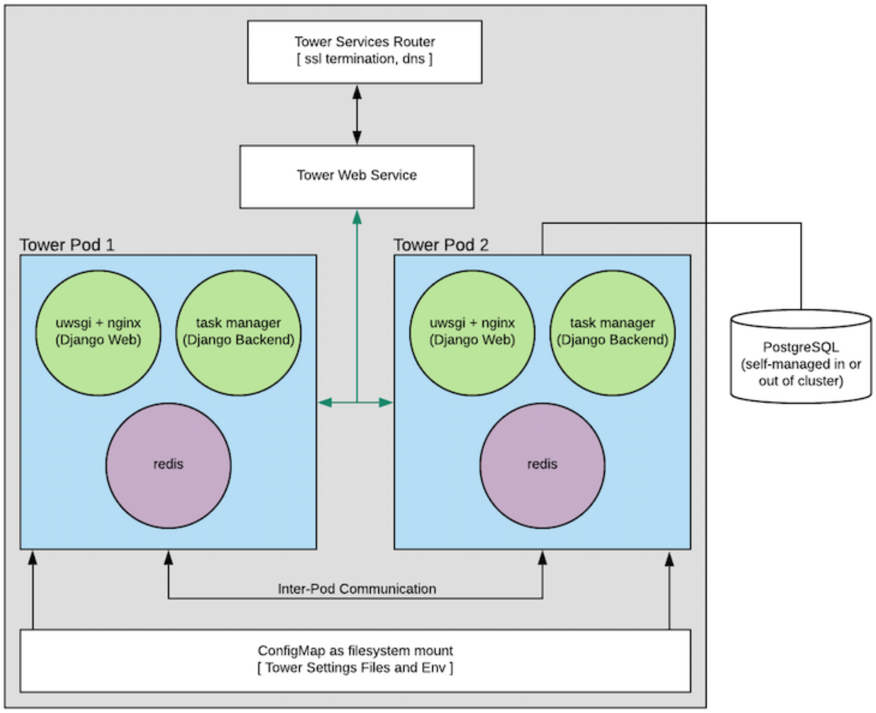
CHAPTER

EIGHT

OPENSHIFT DEPLOYMENT AND CONFIGURATION

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

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


Tower Services Router
[ssl termination, dns]
Tower Web Service
Tower Pod 1
Tower Pod 2
uwsgi + nginx (Django Web)
task manager (Django Backend)
redis
uwsgi + nginx (Django Web)
task manager (Django Backend)
redis
PostgreSQL (self-managed in or out of cluster)
Inter-Pod Communication
ConfigMap as filesystem mount
[Tower Settings Files and Env]

8.1 Tower and OpenShift Basics

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

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

8.2 Configuration Options

Requirements

- Latest supported OpenShift versions 3.x and 4.x (see Red Hat Container Registry Authentication for detail)
- **Per pod default resource requirements:**
  - 6GB RAM
  - 3CPU cores
- Openshift command-line tool (oc) on the machine running the installer
- A setup and running Openshift cluster
- Admin privileges for the account running the openshift installer (`cluster-admin` role is required)

8.3 Basic Configuration

An OpenShift install requires the following parameters to be set:

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

- pg_password

For OpenShift install method, the settings are the same as the traditional Tower install method, except:

- SSL termination for the Tower UI and API is handled through the OpenShift service object. The certificates used here will be generated and signed by the OpenShift internal CA.
- The containerized PostgreSQL pod optionally deployed to OpenShift installs cannot be configured for SSL. If you want SSL-enabled PostgreSQL in an OpenShift environment, you must deploy your PostgreSQL server separately, and configure the Tower nodes (using pg_sslmode).

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

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

OR

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

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

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

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

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

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

The default type for the Tower web service is NodePort. To customize this, set kubernetes_web_svc_type to ClusterIP or LoadBalancer in your inventory file, or pass it as an extra var.

8.4 Resource Requests and Request Planning

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

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

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

Each Tower pod is made up of 3 containers (see *diagram*), each container is configured with a conservative default, but taken all together they can be somewhat substantial. These defaults are also configurable but it's helpful to know what effect that has on the Tower cluster.

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

```
task_cpu_request=1500
```

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

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

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

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

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

The other value that can be tuned:

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

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

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

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

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

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

- 3 CPU Cores
- 6 GB memory

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

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

8.5 Database Configuration and Usage

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

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

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

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

8.6 Backup and Restore

The process for backup and restore resembles that of traditional Tower. From the root of the installer directory of the current Tower version, run:

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

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

8.7 Upgrading

To upgrade a Tower deployment in OpenShift, you need to download and use the most recent installer from http://releases.ansible.com/ansible-tower/setup_openshift. Expect some downtime, just as traditional Tower installations.

8.8 Migrating

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

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

Warning: Do not use special characters in `pg_password` as it may cause the setup to fail.

8.9 Build custom virtual environments

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

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

```
FROM registry.redhat.io/ansible-tower-38/ansible-tower-rhel7
USER root
RUN yum install -y gcc python-devel openssl-devel
RUN umask 0022
RUN virtualenv /var/lib/awx/venv/ansible2.7
RUN /var/lib/awx/venv/ansible2.7/bin/pip install psutil python "ansible==2.7.9"
```

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

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

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

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

If mirroring the vanilla Red Hat images:

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

8.10 Configure TLS with Ansible Tower OpenShift Installer

Configuring TLS for with the Ansible Tower 3.8.x OpenShift installer is done by editing the inventory file, or creating a separate `vars.yml` file, which can be specified when running `./setup_openshift.sh`. In Tower 3.8.x, after running the installer, an additional step is required to set up TLS in order to create a secure Route. The changes needed to the Route and Service objects for your deployment of Tower are described below and can be done from the YAML section in the UI for each resource, or from the command line using `oc edit <resource> <resource-name>`.

1. In the inventory file for the OpenShift installer, before running `./openshift_installer.sh`, set `kubernetes_web_svc_type = "ClusterIP"`. For example, the entry in your inventory file should look like this:

   ```
   kubernetes_web_svc_type: "ClusterIP"
   ```

2. Because the route already exists, we recommend that you modify it as needed (mainly to add the certs). To do this, edit the existing `ansible-tower-web-svc` Route resource by setting the termination to `Edge`. This may be different for your environment, refer to the OpenShift documentation, Creating an edge route with a custom certificate to figure out which is best for you.

3. Create a TLS cert and key then register them with your Certificate Authority.

4. Use that cert and key under the `spec.tls.cert` and `spec.tls.key` sections of the Route YAML similar to the example below.

```
kind: Route
apiVersion: route.openshift.io/v1
metadata:
  name: ansible-tower-web-svc
  namespace: my-namespace
  labels:
    name: ansible-tower-web-svc
spec:
  host: ansible-tower-web-svc-ca-tower.ocp3.ansible.eng.rdu2.redhat.com
  to:
    kind: Service
    name: ansible-tower-web-svc
    weight: 100
  port:
```

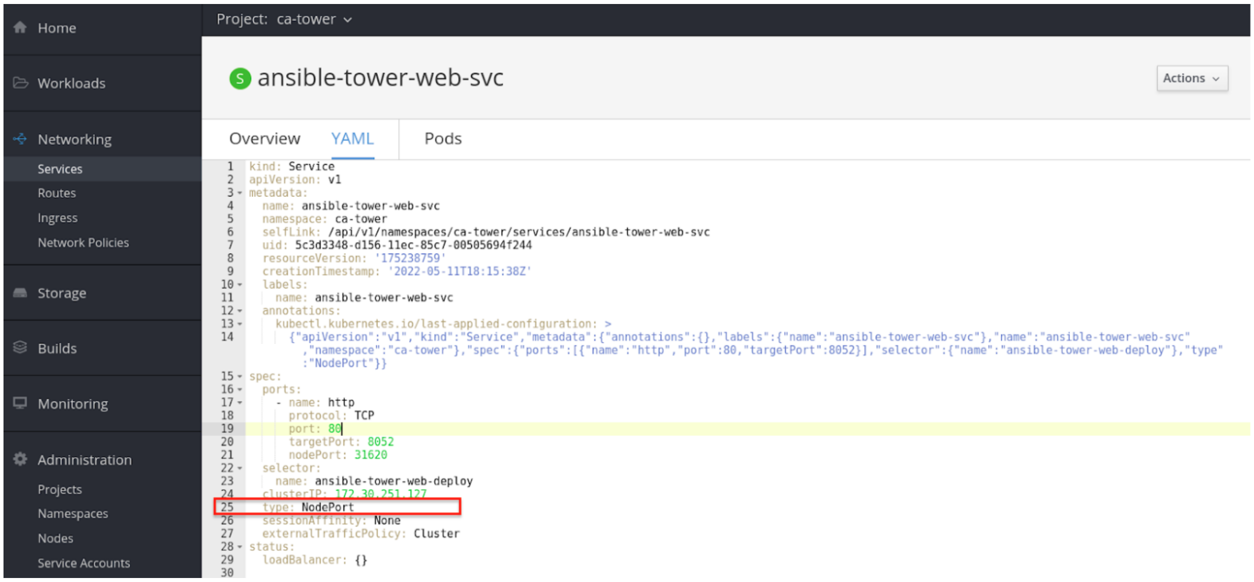
(continues on next page)

(continued from previous page)

```
    targetPort: http
  tls:
    termination: edge
    certificate: |
        -----BEGIN CERTIFICATE-----
        ...
        -----END CERTIFICATE-----

    key: |
        -----BEGIN RSA PRIVATE KEY-----
        ...
        -----END RSA PRIVATE KEY-----
  wildcardPolicy: None
```

5. An unaltered installation of Ansible Tower on OpenShift contains a Service with `service_type = NodePort` similar to the sample YAML below.



```
kind: Service
apiVersion: v1
metadata:
  name: ansible-tower-web-svc
  namespace: ca-tower
  selfLink: /api/v1/namespaces/ca-tower/services/ansible-tower-web-svc
  uid: 5c3d3348-d156-11ec-85c7-00505694f244
  resourceVersion: '175238759'
  creationTimestamp: '2022-05-11T18:15:38Z'
  labels:
    name: ansible-tower-web-svc
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: >
      {"apiVersion":"v1","kind":"Service","metadata":{"annotations":{},"labels":{"name":"ansible-tower-web-svc"},"name":"ansible-tower-web-svc"
        ,"namespace":"ca-tower"},"spec":{"ports":[{"name":"http","port":80,"targetPort":8052}],"selector":{"name":"ansible-tower-web-deploy"},"type"
        :"NodePort"}}
spec:
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 8052
      nodePort: 31620
  selector:
    name: ansible-tower-web-deploy
  clusterIP: 172.30.251.127
  type: NodePort
  sessionAffinity: None
  externalTrafficPolicy: Cluster
status:
  loadBalancer: {}
```

This is the Route prior to selecting *edge* as the type instead of *passthrough*:

```
spec:
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 8052
      nodePort: 31620
  selector:
    name: ansible-tower-web-deploy
  clusterIP: 172.30.251.127
  type: NodePort
  sessionAffinity: None
  externalTrafficPolicy: Cluster
```

6. Optionally, if you do not want to re-run the OpenShift installer with `kubernetes_web_svc_type` set in the inventory file, you can patch the Service resource manually by changing the *type* from `NodePort` to `ClusterIP` on this Service object and remove the `spec.ports[0].nodePort` line, as shown in the example below.

```
spec:
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 8052
  selector:
    name: ansible-tower-web-deploy
  clusterIP: 172.30.251.127
  type: ClusterIP
  sessionAffinity: None
  externalTrafficPolicy: Cluster
```

CHAPTER

NINE

EXECUTION ENVIRONMENTS

Tower allows you to execute jobs via ansible playbook runs directly on a member of the cluster or on a pre-provisioned isolated node. In Ansible Tower 3.8, you can execute jobs in a container group only as-needed per playbook. For more information, see *Container Groups* towards the end of this section.

9.1 Instance Groups

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

- Organizations
- Inventories
- Job Templates

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

Here are some of the things to consider when working with instance groups:

- You may optionally define other groups and group instances in those groups. These groups should be prefixed with `instance_group_`. Instances are not required to be in the tower group alongside other `instance_group_` groups, but one instance **must** be present in the `tower` group. Technically, `tower` is a group like any other `instance_group_` group, but it must always be present, and if a specific group is not associated with a specific resource, then job execution will always fall back to the `tower` group. The `tower` instance group always exists (it cannot be deleted nor renamed).
- Do not create a group named `instance_group_tower`.
- Do not name any instance the same as a group name.

9.1.1 Configuring Instance Groups from the API

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

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

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

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

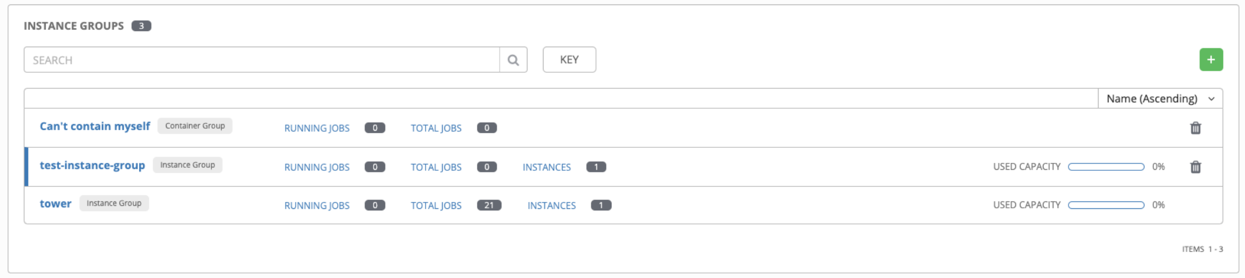
9.1.2 Instance group policies

You can configure Tower instances to automatically join Instance Groups when they come online by defining a policy. These policies are evaluated for every new instance that comes online.

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

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

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



Can't contain myself	Container Group	RUNNING JOBS 0	TOTAL JOBS 0					
test-instance-group	Instance Group	RUNNING JOBS 0	TOTAL JOBS 0	INSTANCES 1	USED CAPACITY	0%		
tower	Instance Group	RUNNING JOBS 0	TOTAL JOBS 21	INSTANCES 1	USED CAPACITY	0%		

9.1.3 Notable policy considerations

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

9.1.4 Manually pinning instances to specific groups

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

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

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

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

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

9.1.5 Job Runtime Behavior

When you run a job associated with a instance group, some behaviors worth noting are:

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

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

9.1.6 Control Where a Job Runs

If any of the job template, inventory, or organization has instance groups associated with them, a job ran from that job template will not be eligible for the default behavior. That means that if all of the instances inside of the instance groups associated with these 3 resources are out of capacity, the job will remain in the pending state until capacity becomes available.

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

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

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

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

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

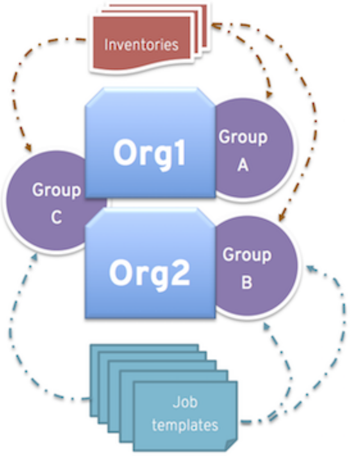
In addition, it is possible to not associate an instance group with one resource but designate another resource as the fallback. For example, not associating an instance group with a job template and have it fall back to the inventory and/or the organization's instance group.

This presents two other great use cases:

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

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

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



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

9.1.7 Deprovision Instance Groups

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

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

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

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

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

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

9.1.8 Isolated Instance Groups

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

```
[tower]
towerA
towerB
towerC

[instance_group_security]
towerB
towerC

[isolated_group_govcloud]
isolatedA
isolatedB

[isolated_group_govcloud:vars]
controller=security
```

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

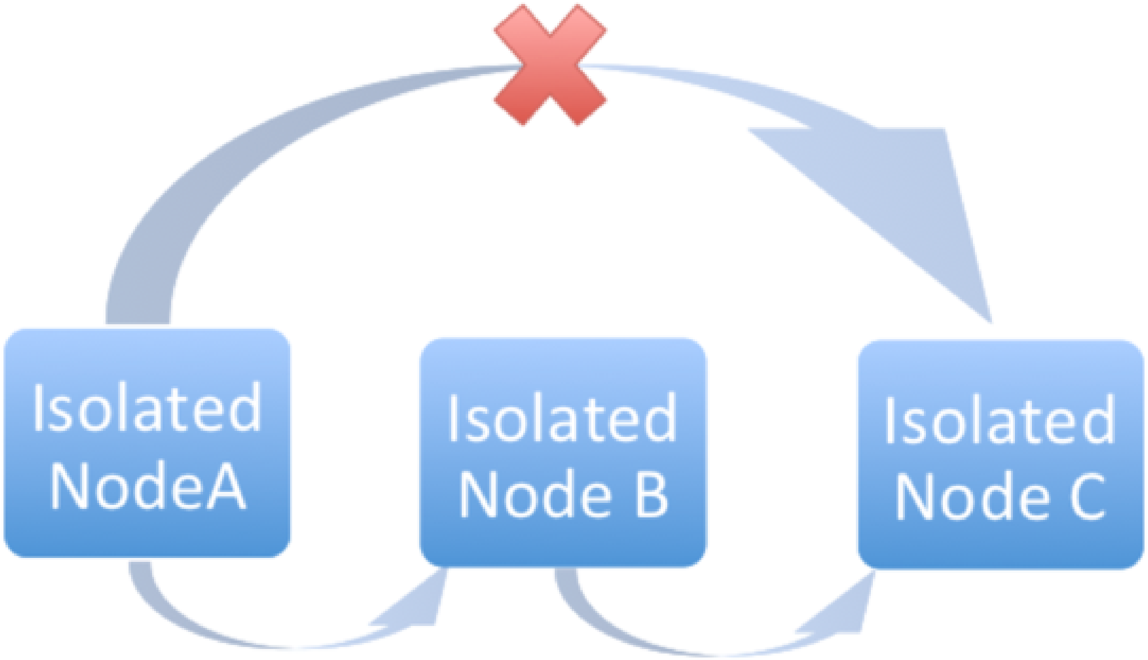
When a job is scheduled to run on an "isolated" instance:

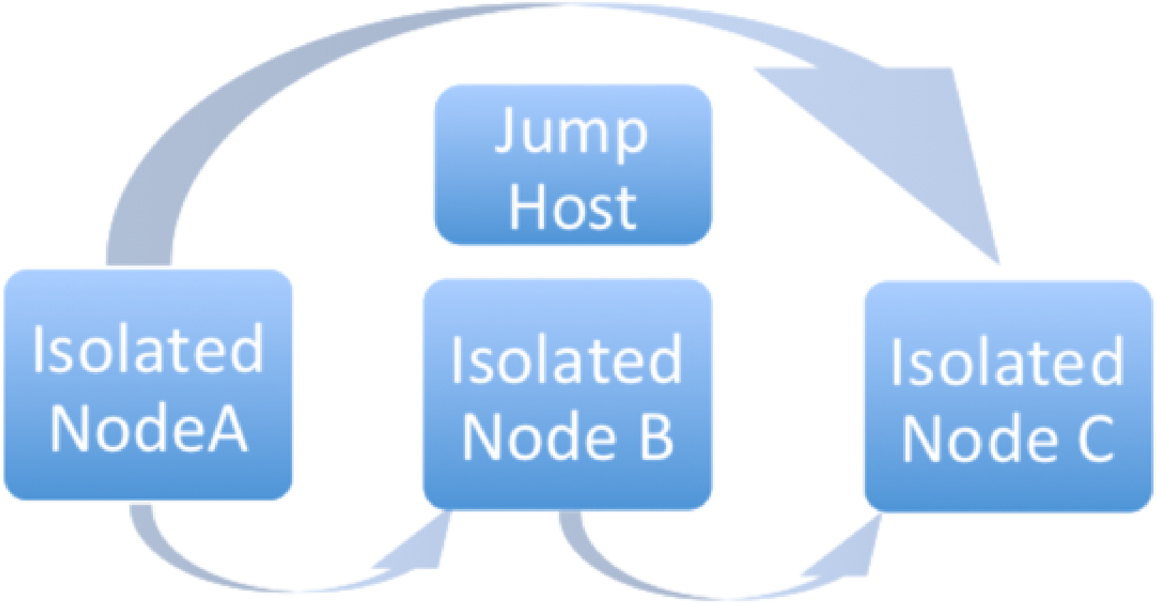
- The "controller" instance compiles metadata required to run the job and copies it to the "isolated" instance.

- Once the metadata has been synchronized to the isolated host, the "controller" instance starts a process on the "isolated" instance, which consumes the metadata and starts running `ansible/ansible-playbook`. As the playbook runs, job artifacts (such as stdout and job events) are written to disk on the "isolated" instance.
- While the job runs on the "isolated" instance, the "controller" instance periodically copies job artifacts (stdout and job events) from the "isolated" instance. It consumes these until the job finishes running on the "isolated" instance.

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

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

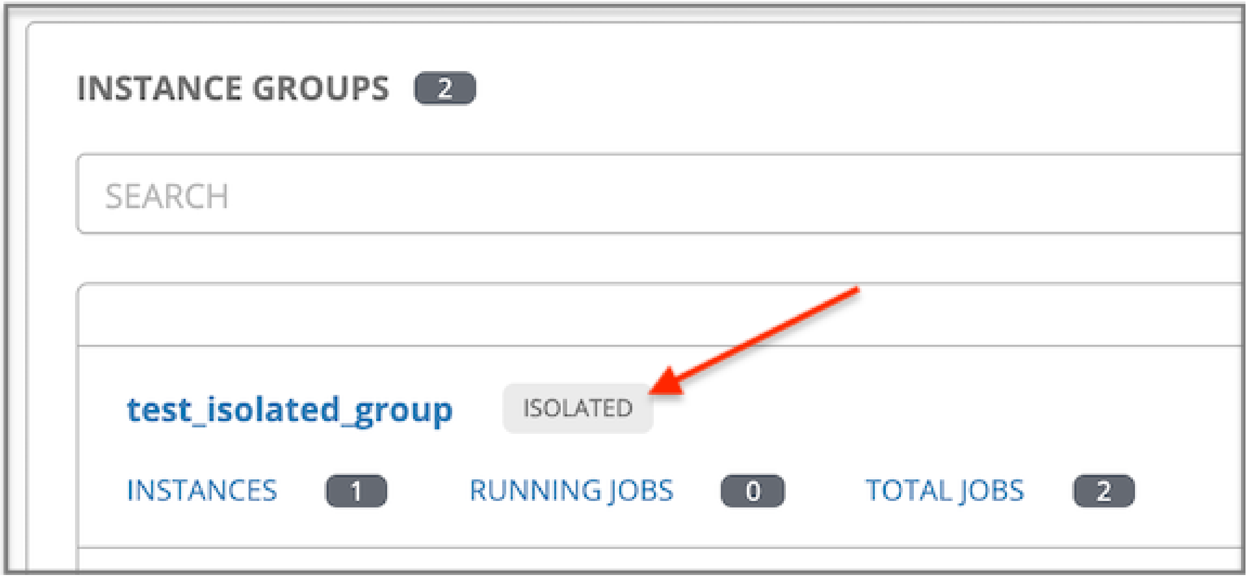




The recommended system configurations with isolated groups are as follows:

- Do not create a group named `isolated_group_tower`.
- Do not put any isolated instances inside the tower group or other ordinary instance groups.
- Define the controller variable as either a group variable or as a host variable on all the instances in the isolated group. Do not allow isolated instances in the same group to have a different value for this variable - the behavior in this case cannot be predicted.
- Do not put an isolated instance in more than one isolated group.
- Do not put an instance in both ordinary groups and isolated groups.
- Isolated instances can be installed on RHEL 7 and later.
- The following durations associated with isolated groups can be configured in the **Jobs** tab of the Settings (⚙) menu:
  - **Isolated Status Check Interval**: 30 seconds is the default amount of time set to sleep between status checks for jobs running on isolated instances.
  - **Isolated Launch Timeout**: 600 seconds (10 mins) is the default timeout for launching jobs on isolated instances. This includes the time needed to copy source control files (playbooks) to the isolated instance.
  - **Isolated Connection Timeout**: 10 seconds is the default Ansible SSH connection timeout when communicating with isolated instances. This value should be substantially greater than the expected network latency.

Isolated groups are labeled accordingly in the Instance Groups list view of the Tower User Interface.



9.2 Container Groups

Note: The Container Groups feature is in tech preview and is subject to change in a future release.

Ansible Tower supports Container Groups, which allow you to execute jobs in Tower regardless of whether Tower is installed as a standalone, in a virtual environment, or in a container. Container groups act as a pool of resources within a virtual environment. You can create instance groups to point to an OpenShift or Kubernetes container, which are job environments that are provisioned on-demand as a Pod that exists only for the duration of the playbook run. This is known as the ephemeral execution model and ensures a clean environment for every job run.

In some cases, it is desirable to have the execution environment be "always-on", which is configured through the creation of an instance.

9.2.1 Create a container group

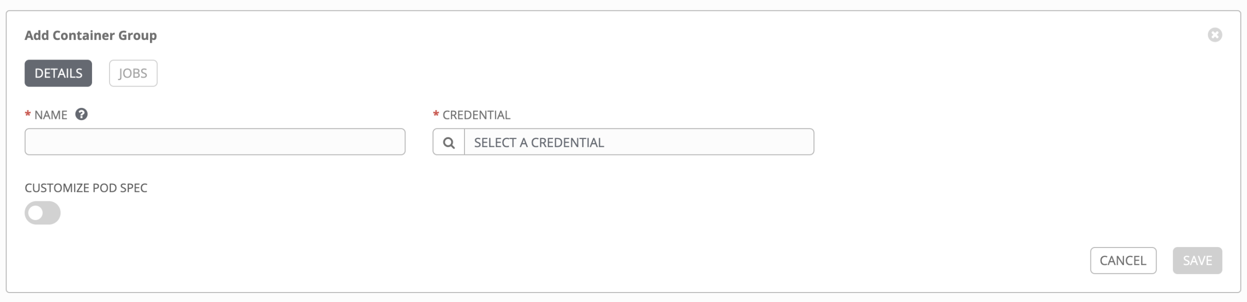
A `ContainerGroup` is simply an `InstanceGroup` that has an associated Credential that allows for connecting to an OpenShift or Kubernetes cluster. To set up a container group on Kubernetes or OpenShift, you must first have the following:

- A namespace you can launch into (there is a "default" namespace but most likely varied by customer)
- A service account that has the roles that allow it to launch and manage Pods in this namespace
- A token associated with that service account (Kubernetes or OpenShift Bearer Token)
- A CA certificate associated with the cluster

To create a container group:

1. Use the Tower User Interface to create an OpenShift or Kubernetes API bearer token credential that will be used with your container group, see Add a New Credential in the *Ansible Tower User Guide* for detail.
2. Create a new container group by navigating to the Instance Groups configuration window 🗄.

3. Click the + button and select **Create Container Group**.

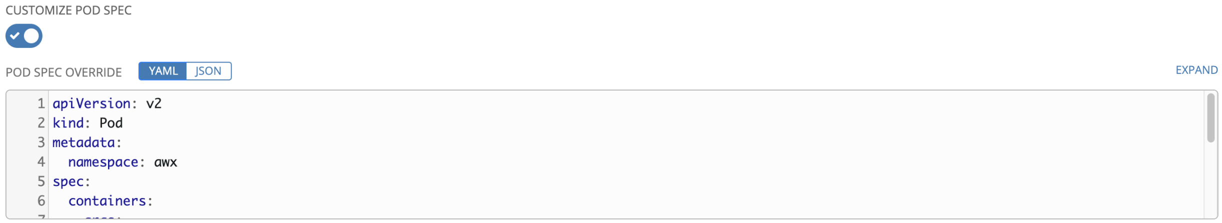


4. Enter a name for your new container group and select the credential previously created to associate it to the container group.

9.2.2 Customize the Pod spec

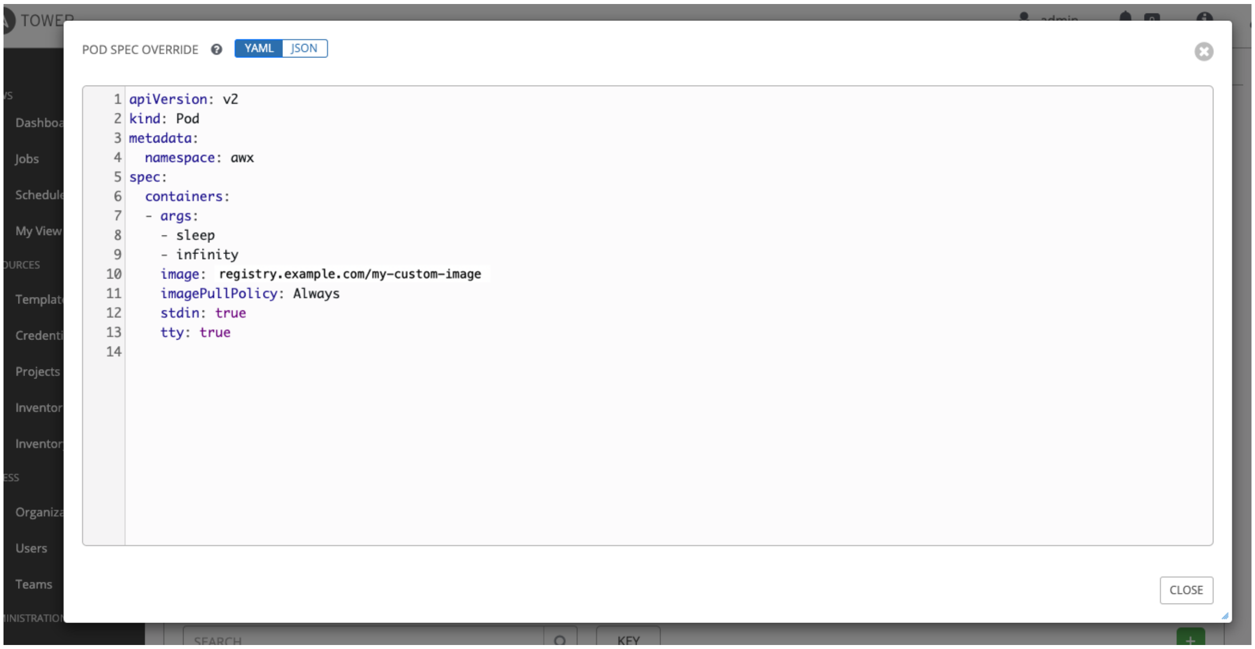
Tower provides a simple default Pod specification, however, you can provide a custom YAML (or JSON) document that overrides the default Pod spec. This field uses any custom fields (i.e., `image` or `namespace`) that can be "serialized" as valid Pod JSON or YAML. A full list of options can be found in the Kubernetes documentation.

To customize the Pod spec, specify the namespace in the **Pod Spec Override** field by using the toggle to enable and expand the **Pod Spec Override** field and click **Save** when done.



```
apiVersion: v2
kind: Pod
metadata:
  namespace: awx
spec:
  containers:
```

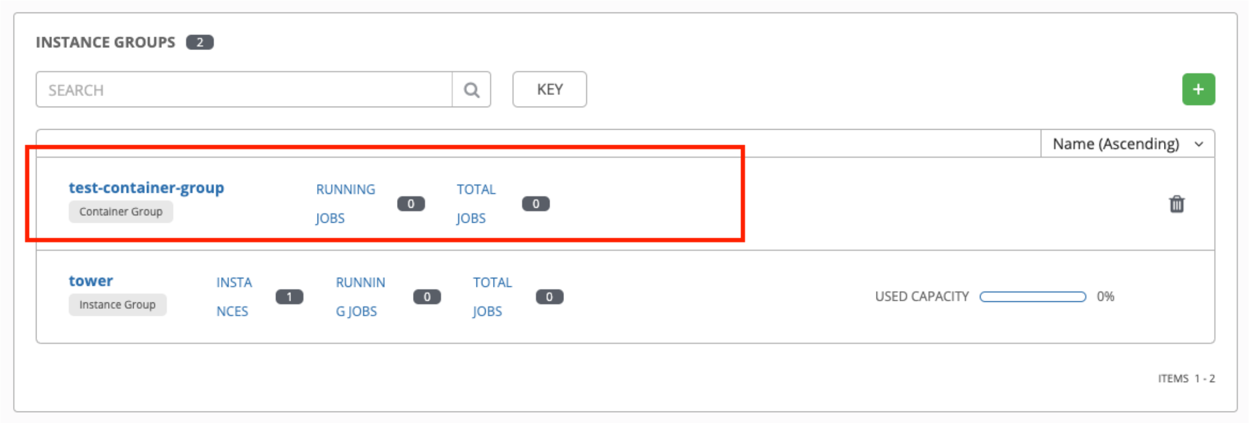
You may provide additional customizations, if needed. Click **Expand** to view the entire customization window.



```yaml
apiVersion: v2
kind: Pod
metadata:
  namespace: awx
spec:
  containers:
  - args:
    - sleep
    - infinity
    image: registry.example.com/my-custom-image
    imagePullPolicy: Always
    stdin: true
    tty: true
```

Note: The default container group image is pulled from `registry.redhat.com`. That means that an image pull secret (`imagePullSecrets`) should be specified in the Pod spec. Refer to the *Allowing Pods to Reference Images from Other Secured Registries* section of the Red Hat Container Registry Authentication article for more information on how to create image pull secrets.

Once the container group is successfully created, the **Details** tab of the newly created container group remains, which allows you to review and edit your container group information. This is the same menu that is opened if the Edit (✎) button is clicked from the **Instance Group** link. You can also edit **Instances** and review **Jobs** associated with this instance group.



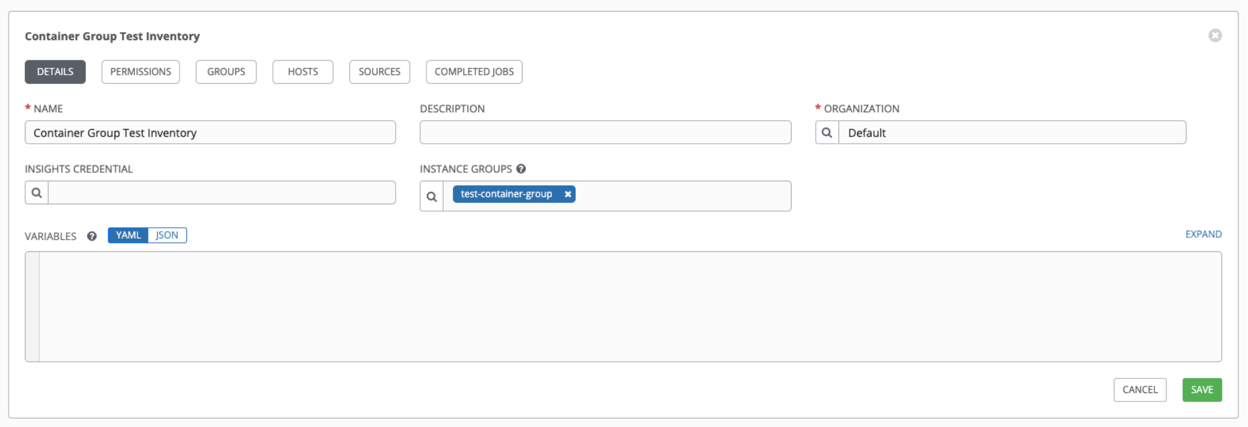
Container groups and instance groups are labeled accordingly.

Note: Despite the fact that customers have custom Pod specs, upgrades may be difficult if the default `pod_spec` changes. Most any manifest can be applied to any namespace, with the namespace specified separately, most likely you will only need to override the namespace. Similarly, pinning a default image for different releases of Tower to different versions of the default job runner container is tricky. If the default image is specified in the Pod spec, then upgrades do not pick up the new default changes are made to the default Pod spec.

9.2.3 Verify container group functions

To verify the deployment and termination of your container:

1. Create a mock inventory and associate the container group to it by populating the name of the container group in the **Instance Group** field. See Add a new inventory in the *Ansible Tower User Guide* for detail.



Container Group Test Inventory

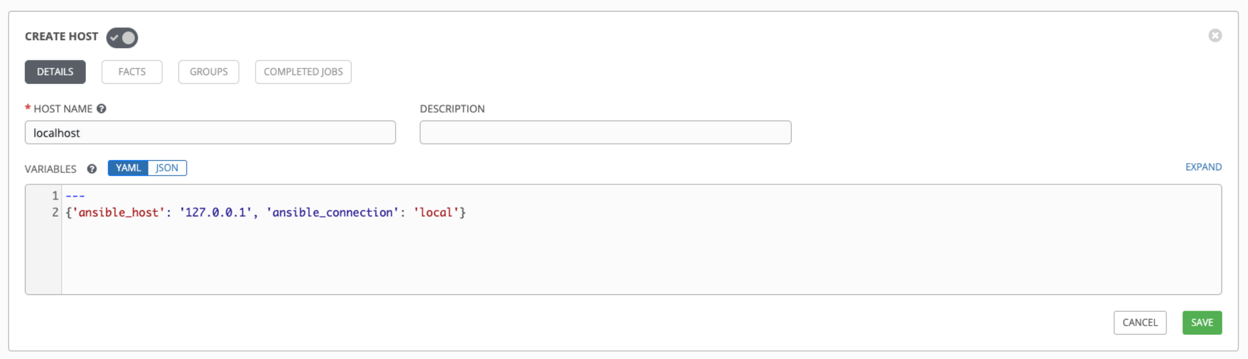
* NAME: Container Group Test Inventory
* ORGANIZATION: Default

INSTANCE GROUPS: test-container-group

VARIABLES: YAML

2. Create "localhost" host in inventory with variables:

```
{'ansible_host': '127.0.0.1', 'ansible_connection': 'local'}
```



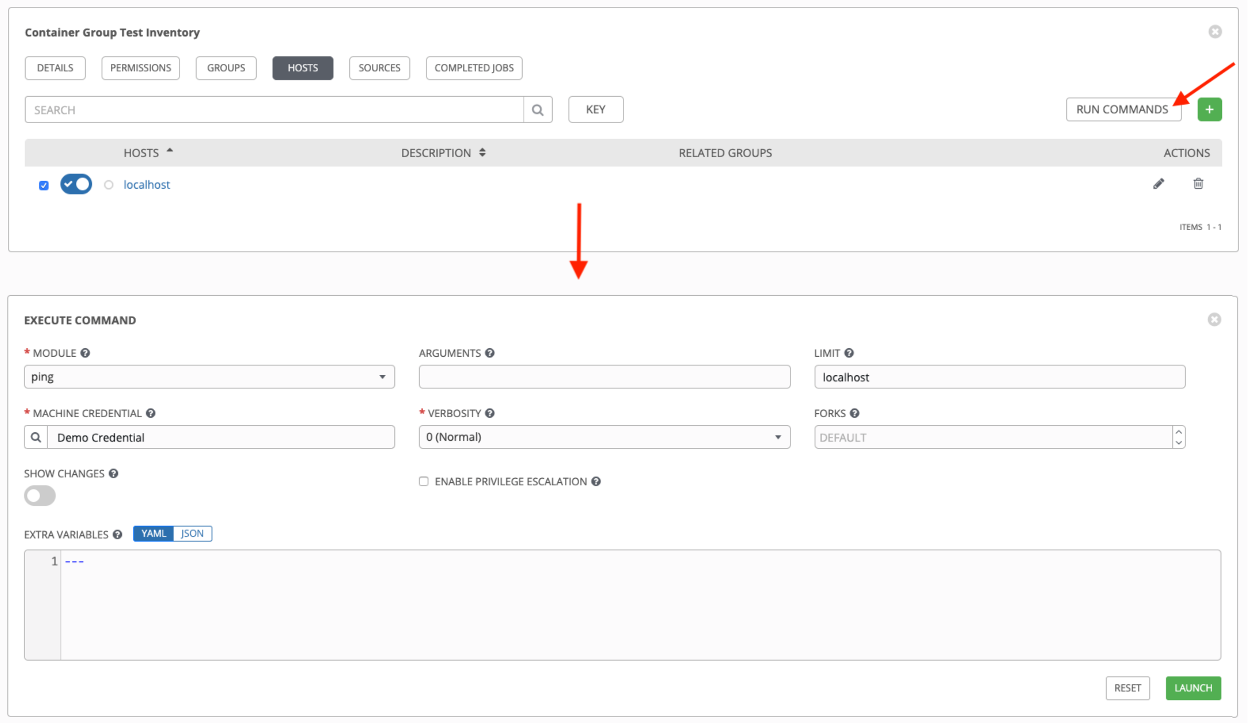
CREATE HOST

* HOST NAME: localhost

VARIABLES: YAML

```
---
{'ansible_host': '127.0.0.1', 'ansible_connection': 'local'}
```

3. Launch an ad hoc job against the localhost using the *ping* or *setup* module. Even though the **Machine Credential** field is required, it does not matter which one is selected for this simple test.



Container Group Test Inventory

☑ localhost

ITEMS 1 - 1

EXECUTE COMMAND

* MODULE: ping

ARGUMENTS:

LIMIT: localhost

* MACHINE CREDENTIAL: Demo Credential

* VERBOSITY: 0 (Normal)

FORKS: DEFAULT

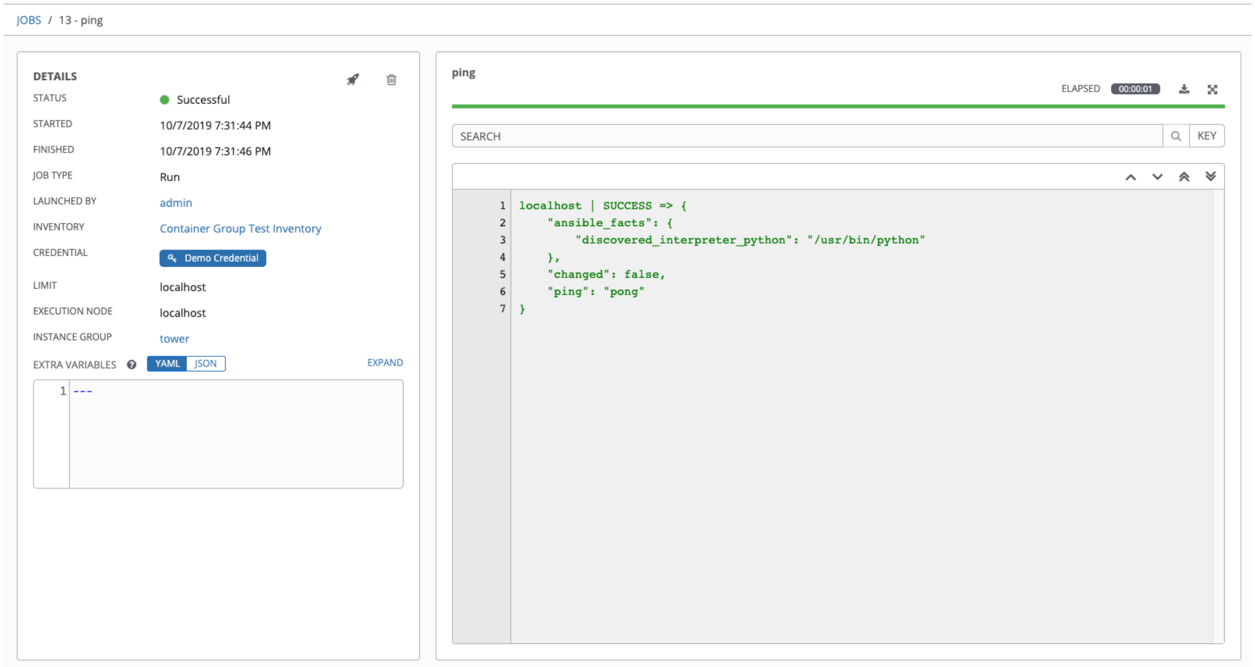
SHOW CHANGES

□ ENABLE PRIVILEGE ESCALATION

EXTRA VARIABLES: YAML JSON

```
---
```

You can see in the jobs detail view the container was reached successfully using one of ad hoc jobs.



JOBS / 13 - ping

DETAILS

STATUS	● Successful
STARTED	10/7/2019 7:31:44 PM
FINISHED	10/7/2019 7:31:46 PM
JOB TYPE	Run
LAUNCHED BY	admin
INVENTORY	Container Group Test Inventory
CREDENTIAL	Demo Credential
LIMIT	localhost
EXECUTION NODE	localhost
INSTANCE GROUP	tower
EXTRA VARIABLES	YAML JSON

```
---
```

ping

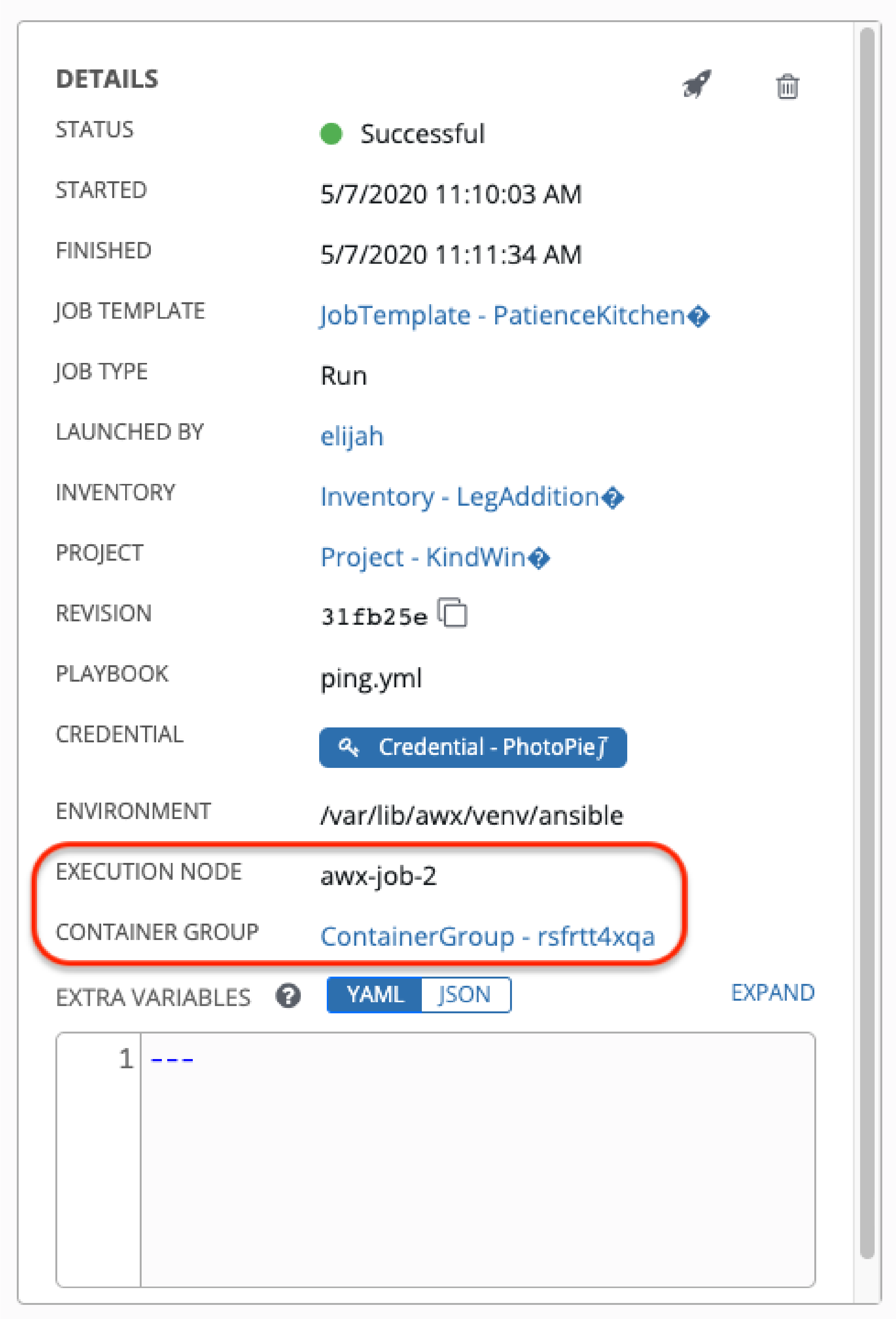
ELAPSED 00:00:01

```
localhost | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

If you have an OpenShift or Kubernetes UI, you can see Pods appear and disappear as they deploy and terminate. Alternatively, you can use the CLI to perform a `get pod` operation on your namespace to watch these same events occurring in real-time.

9.2.4 View container group jobs

When you run a job associated with a container group, you can see the details of that job in the **Details** view and its associated Instance Group and the execution node that spun up.



DETAILS

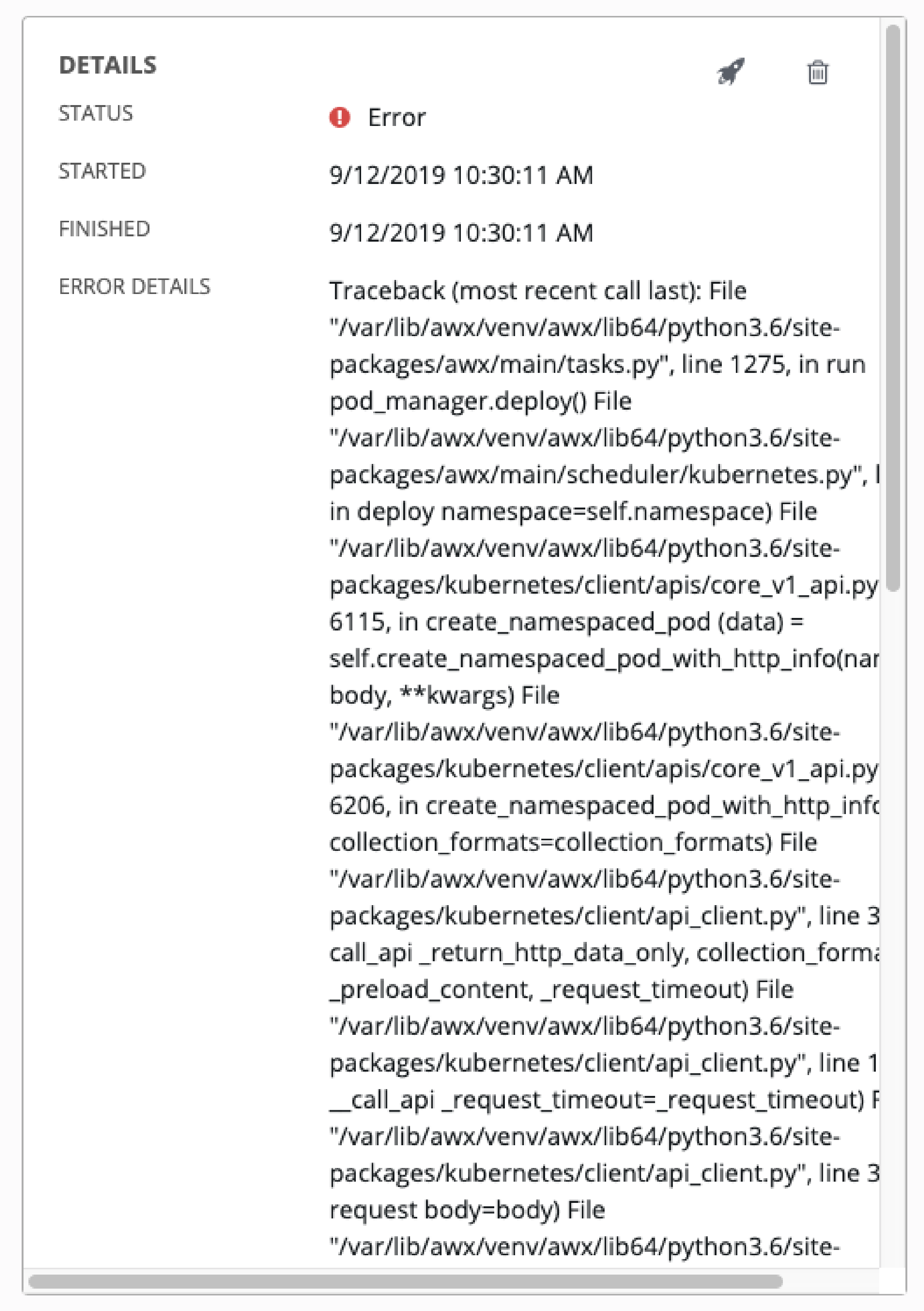
STATUS	● Successful
STARTED	5/7/2020 11:10:03 AM
FINISHED	5/7/2020 11:11:34 AM
JOB TEMPLATE	JobTemplate - PatienceKitchen�
JOB TYPE	Run
LAUNCHED BY	elijah
INVENTORY	Inventory - LegAddition�
PROJECT	Project - KindWin�
REVISION	`31fb25e`
PLAYBOOK	ping.yml
CREDENTIAL	Credential - PhotoPie∫
ENVIRONMENT	/var/lib/awx/venv/ansible
EXECUTION NODE	awx-job-2
CONTAINER GROUP	ContainerGroup - rsfrtt4xqa

EXTRA VARIABLES

```
---
```

9.2.5 Kubernetes failure conditions

When running a container group and Kubernetes responds that the resource quota has been exceeded, Tower keeps the job in pending state. Other failures result in the traceback of the **Error Details** field showing the failure reason, similar to the example here:



DETAILS

STATUS ❗ Error

STARTED 9/12/2019 10:30:11 AM

FINISHED 9/12/2019 10:30:11 AM

ERROR DETAILS

```
Traceback (most recent call last): File
"/var/lib/awx/venv/awx/lib64/python3.6/site-
packages/awx/main/tasks.py", line 1275, in run
pod_manager.deploy() File
"/var/lib/awx/venv/awx/lib64/python3.6/site-
packages/awx/main/scheduler/kubernetes.py", l
in deploy namespace=self.namespace) File
"/var/lib/awx/venv/awx/lib64/python3.6/site-
packages/kubernetes/client/apis/core_v1_api.py
6115, in create_namespaced_pod (data) =
self.create_namespaced_pod_with_http_info(nar
body, **kwargs) File
"/var/lib/awx/venv/awx/lib64/python3.6/site-
packages/kubernetes/client/apis/core_v1_api.py
6206, in create_namespaced_pod_with_http_infc
collection_formats=collection_formats) File
"/var/lib/awx/venv/awx/lib64/python3.6/site-
packages/kubernetes/client/api_client.py", line 3
call_api _return_http_data_only, collection_forma
_preload_content, _request_timeout) File
"/var/lib/awx/venv/awx/lib64/python3.6/site-
packages/kubernetes/client/api_client.py", line 1
__call_api _request_timeout=_request_timeout) F
"/var/lib/awx/venv/awx/lib64/python3.6/site-
packages/kubernetes/client/api_client.py", line 3
request body=body) File
"/var/lib/awx/venv/awx/lib64/python3.6/site-
```

9.2.6 Container capacity limits

Capacity limits and quotas for containers are defined via objects in the Kubernetes API:

- To set limits on all pods within a given namespace, use the `LimitRange` object. Refer to the Kubernetes documentation for Configure Default Memory Requests and Limits for a Namespace.
- To set limits directly on the pod definition launched by Tower, see *Customize the Pod spec* and refer to the Kubernetes documentation to set the options to Assign Memory Resources to Containers and Pods.

Note: Container groups do not use the capacity algorithm that normal nodes use. You would need to explicitly set the number of forks at the job template level, for instance. If forks are configured in Tower, that setting will be passed along to the container.

CHAPTER

TEN

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:** captures callback receiver logs that handles callback events when running ansible jobs.
- **dispatcher.log:** captures log messages for the Tower dispatcher worker service.
- **management_playbooks.log:** captures the logs of management playbook runs, the isolated job runs like copying the metadata, etc.
- **rsyslog.err:** captures rsyslog errors authenticating with external logging services when sending logs to them.
- **task_system.log:** - captures the logs of tasks that Tower is running in the background, such as adding tower cluster instances and logs related to information gathering/processing for analytics, etc.
- **tower.log:** captures the log messages such as runtime errors that occur when the job is executed.
- **tower_rbac_migrations.log:** captures the logs for rbac database migration or upgrade.
- **tower_system_tracking_migrations.log:** captures the logs tower system tracking migration or upgrade.
- **wsbroadcast.log:** captures the logs of websocket connections in the tower nodes.
- **isolated_manager.log:** captures logs associated with isolated nodes.

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

- **awx-callback-receiver.log:** captures the log of callback receiver that handles callback events when running ansible jobs, managed by supervisord.
- **awx-daphne.log:** captures the logs of Websocket communication of WebUI.
- **awx-dispatcher.log:** captures the logs that occur when dispatching a task to a tower instance, such as when running a job.
- **awx-rsyslog.log:** captures the logs for rsyslog service.
- **awx-uwsgi.log:** captures the logs related to uWSGI, which is an application server.
- **awx-wsbroadcast.log:** captures the logs of websocket service that is used by tower.
- **failure-event-handler.stderr.log:** captures the standard errors for /usr/bin/failure-event-handler supervisord's subprocess.
- **supervisord.log:** captures the logs related to supervisord itself.

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

You can expect the following log paths to be generated by services used by Tower (and Ansible Automation Platform):

- **/var/log/nginx/**
- **/var/opt/rh/rh-postgresql10/**
- **/var/opt/rh/rh-redis5/log/redis/**

```
"Mooving around: Consolidated logfiles for easier access!"
    \
     \    ^__^
      \   (oo)\_______
          (__)\       )\/\
              ||----w |
              ||     ||
```

CHAPTER

ELEVEN

TOWER LOGGING AND AGGREGATION

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

Installing Ansible Tower will install a newer version of rsyslog, which will replace the version that comes with the RHEL base. The version of rsyslog that is installed by Ansible Tower does not include the following rsyslog modules:

- rsyslog-udpspoof.x86_64
- rsyslog-libdbi.x86_64

After installing Ansible Tower, use only the Tower provided rsyslog package for any logging outside of Tower that may have previously been done with the RHEL provided rsyslog package. If you already use rsyslog for logging system logs on Tower instances, you can continue to use rsyslog to handle logs from outside of Tower by running a separate rsyslog process (using the same version of rsyslog that Tower is), and pointing it to a separate /etc/rsyslog.conf.

Note: For systems that use rsyslog outside of Tower (on the Tower VM/machine), consider any conflict that may arise with also using new version of rsyslog that comes with Tower.

You can configure from the `/api/v2/settings/logging/` endpoint how the Tower rsyslog process handles messages that have not yet been sent in the event that your external logger goes offline:

- `LOG_AGGREGATOR_MAX_DISK_USAGE_GB`: specifies the amount of data to store (in gigabytes) during an outage of the external log aggregator (defaults to 1). Equivalent to the `rsyslogd queue.maxdiskspace` setting.
- `LOG_AGGREGATOR_MAX_DISK_USAGE_PATH`: specifies the location to persist logs that should be retried after an outage of the external log aggregator (defaults to `/var/lib/awx`). Equivalent to the `rsyslogd queue.spoolDirectory` setting.

For example, if Splunk goes offline, rsyslogd stores a queue on the disk until Splunk comes back online. By default, it will store up to 1GB of events (while Splunk is offline) but you can make that more than 1GB if necessary, or change the path where you save the queue.

11.1 Loggers

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

- `job_events`: Provides data returned from the Ansible callback module
- `activity_stream`: Displays the record of changes to the objects within the Ansible Tower application
- `system_tracking`: Provides fact data gathered by Ansible `setup` module (i.e. `gather_facts: True`) when job templates are ran with **Enable Fact Cache** selected
- `awx`: Provides generic server logs, which include logs that would normally be written to a file. It contains the standard metadata that all logs have, except it only has the message from the log statement.

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

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

To configure various logging components in Ansible Tower, select **System** from the (⚙) menu located on the left navigation bar.

11.1.1 Log message schema

Common schema for all loggers:

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

11.1.2 Activity stream schema

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

11.1.3 Job event schema

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

This logger also includes the common fields.

11.1.4 Scan / fact / system tracking data schema

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

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

11.1.5 Job status changes

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

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

11.1.6 Tower logs

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

11.1.7 Logging Aggregator Services

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

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

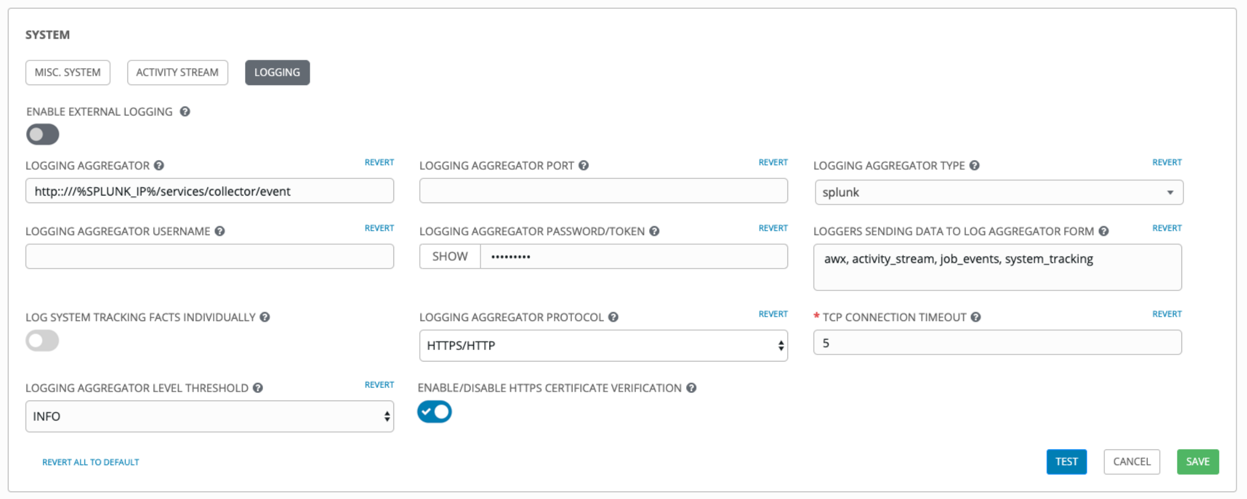
Splunk

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

```
https://yourtowerfqdn.com/api/v2/settings/logging

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

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


SYSTEM

ENABLE EXTERNAL LOGGING

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

LOGGING AGGREGATOR PORT:

LOGGING AGGREGATOR TYPE: splunk

LOGGING AGGREGATOR USERNAME:

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

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

LOG SYSTEM TRACKING FACTS INDIVIDUALLY

LOGGING AGGREGATOR PROTOCOL: HTTPS/HTTP

* TCP CONNECTION TIMEOUT: 5

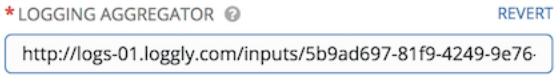
LOGGING AGGREGATOR LEVEL THRESHOLD: INFO

ENABLE/DISABLE HTTPS CERTIFICATE VERIFICATION

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

Loggly

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

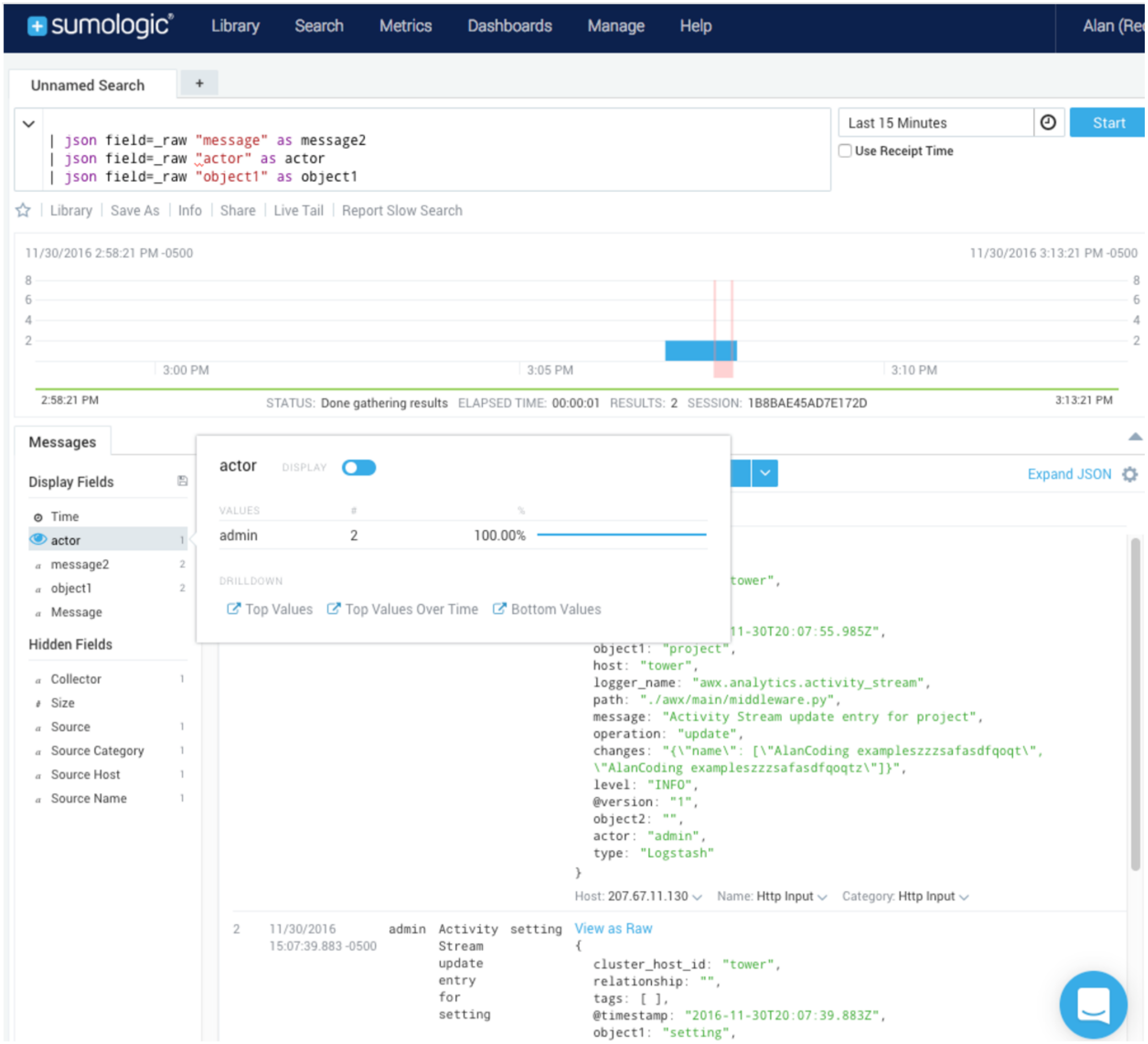


* LOGGING AGGREGATOR

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

Sumologic

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



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

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

VALUES	#	%
admin	2	100.00%

```
tower",

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

Host: 207.67.11.130 Name: Http Input Category: Http Input

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

Elastic stack (formerly ELK stack)

Standing up your own version the elastic stack requires no modification to the logstash `logstash.conf` file.

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

11.2 Set Up Logging with Tower

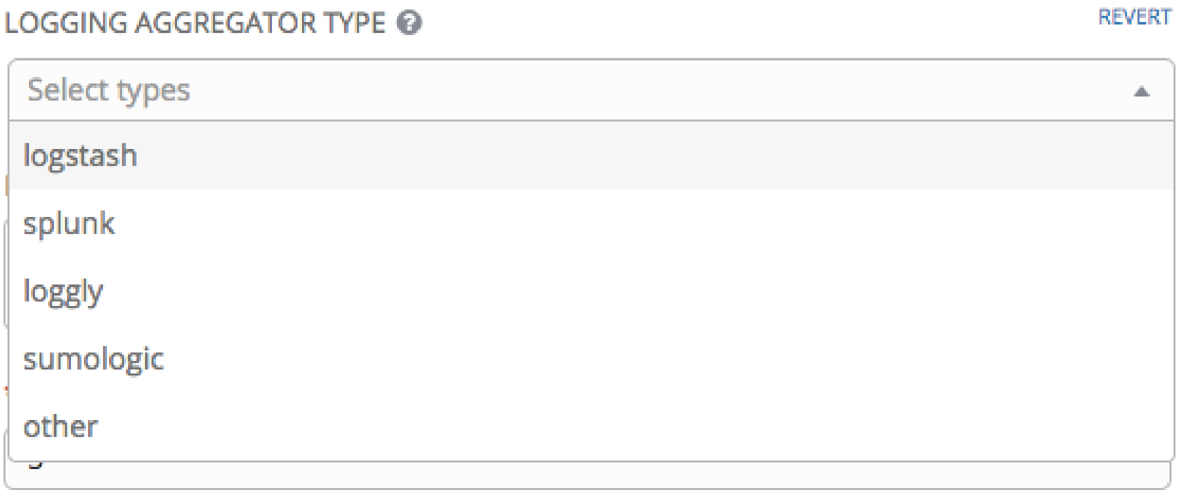
To set up logging to any of the aggregator types:

1. Click the Settings (⚙) icon from the left navigation bar.
2. Select **System**.
3. In the System screen, select the **Logging** tab.
4. Set the configurable options from the fields provided:

- **Enable External Logging**: Click the toggle button to **ON** if you want to send logs to an external log aggregator.
- **Logging Aggregator**: Enter the hostname or IP address you want to send logs.
- **Logging Aggregator Port**: Specify the port for the aggregator if it requires one.

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

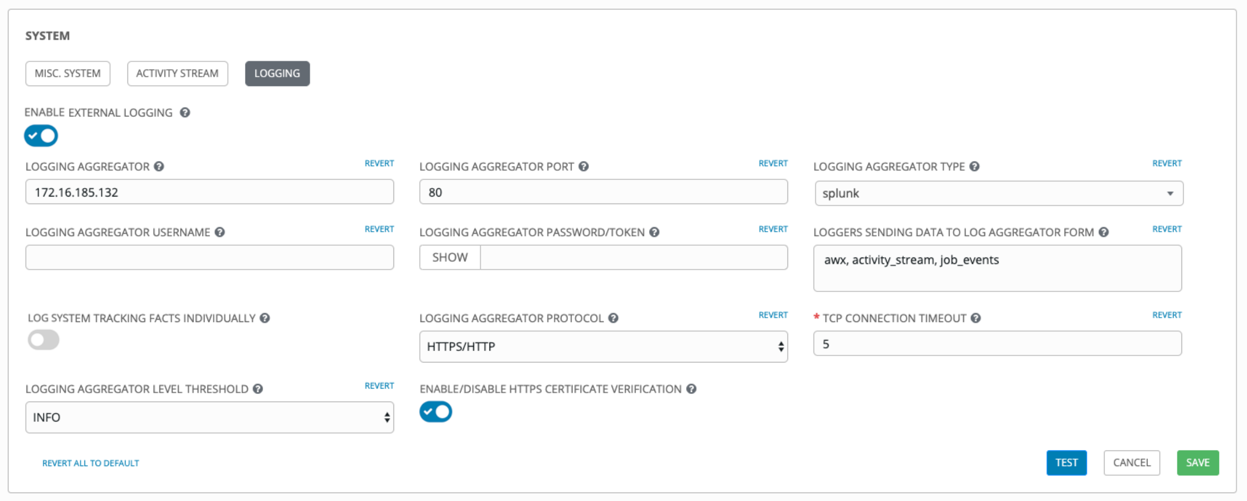
- **Logging Aggregator Type**: Click to select the aggregator service from the drop-down menu:



- **Logging Aggregator Username**: Enter the username of the logging aggregator if it requires it.

- **Logging Aggregator Password/Token**: Enter the password of the logging aggregator if it requires it.
- **Loggers to Send Data to the Log Aggregator Form**: All four types of data are pre-populated by default. Click the tooltip ? icon next to the field for additional information on each data type. Delete the data types you do not want.
- **Log System Tracking Facts Individually**: Click the tooltip ? icon for additional information whether or not you want to turn it on, or leave it off by default.
- **Logging Aggregator Protocol**: Click to select a connection type (protocol) to communicate with the log aggregator. Subsequent options vary depending on the selected protocol.
- **TCP Connection Timeout**: Specify the connection timeout in seconds. This option is only applicable to HTTPS and TCP log aggregator protocols.
- **Logging Aggeregator Level Threshold**: Select the level of severity you want the log handler to report.
- **Enable/Disable HTTPS Certificate Verification**: Certificate verification is enabled by default for HTTPS log protocol. Click the toggle button to **OFF** if you do not want the log handler to verify the HTTPS certificate sent by the external log aggregator before establishing a connection.

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



SYSTEM

ENABLE EXTERNAL LOGGING: ON

LOGGING AGGREGATOR: 172.16.185.132

LOGGING AGGREGATOR PORT: 80

LOGGING AGGREGATOR TYPE: splunk

LOGGING AGGREGATOR USERNAME:

LOGGING AGGREGATOR PASSWORD/TOKEN: SHOW

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

LOG SYSTEM TRACKING FACTS INDIVIDUALLY: OFF

LOGGING AGGREGATOR PROTOCOL: HTTPS/HTTP

* TCP CONNECTION TIMEOUT: 5

LOGGING AGGREGATOR LEVEL THRESHOLD: INFO

ENABLE/DISABLE HTTPS CERTIFICATE VERIFICATION: ON

7. When done, click **Save** to apply the settings or **Cancel** to abandon the changes.
8. To verify if your configuration is set up correctly, click **Save** first then click **Test**. This sends a test log message to the log aggregator using the current logging configuration in Ansible Tower. You should check to make sure this test message was received by your external log aggregator.

Note: If the **Test** button is disabled, it is an indication that the fields are different than their initial values so save your changes first, and make sure the **Enable External Logging** toggle is set to ON.

11.3 Troubleshooting Logging with Tower

If you have sent a message with the test button to your configured logging service via http/https, but did not receive the message, check the `/var/log/tower/rsyslog.err` log file. This is where errors are stored if they occurred when authenticating rsyslog with an http/https external logging service. Note that if there are no errors, this file will not exist.

CHAPTER

TWELVE

METRICS

A metrics endpoint is available in the API: /api/v2/metrics/ that surfaces instantaneous metrics about Tower, which can be consumed by system monitoring software like the open source project Prometheus.

The type of data shown at the metrics/ endpoint is Content-type: text/plain and application/json as well. This endpoint contains useful information, such as counts of how many active user sessions there are, or how many jobs are actively running on each Tower node. Prometheus can be configured to scrape these metrics from Tower by hitting the Tower metrics endpoint and storing this data in a time-series database. Clients can later use Prometheus in conjunction with other software like Grafana or Metricsbeat to visualize that data and set up alerts.

12.1 Set up Prometheus

To set up and use Prometheus, you will need to install Prometheus on a virtual machine or container. Refer to the Prometheus documentation for further detail.

1. In the Prometheus config file (typically prometheus.yml), specify a <token_value>, a valid user/password for a Tower user you have created, and a <tower_host>.

   Note: Alternatively, you can provide an OAuth2 token (which can be generated at /api/v2/users/N/personal_tokens/). By default, the config assumes a user with username=admin and password=password.

   Using an OAuth2 Token, created at the /api/v2/tokens endpoint to authenticate prometheus with Tower, the following example provides a valid scrape config if the URL for your Tower's metrics endpoint was https://tower_host:443/metrics.

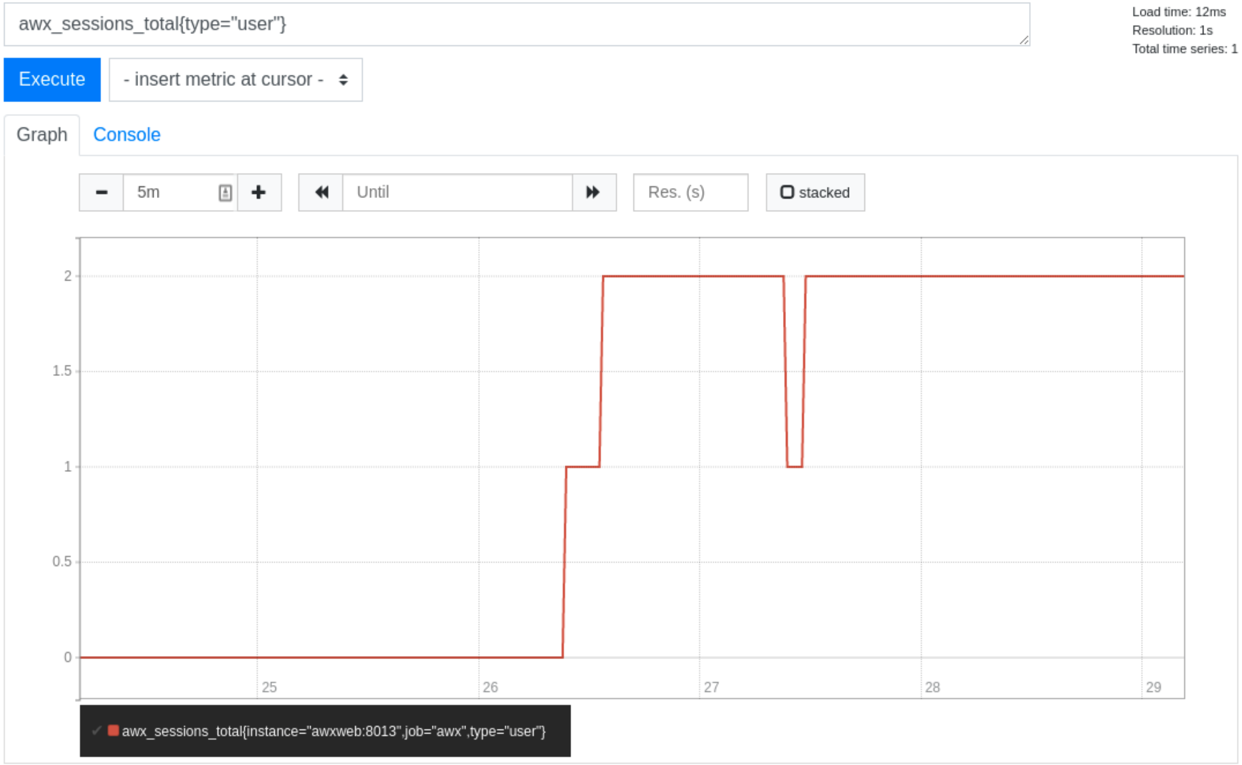
```
scrape_configs

  - job_name: 'tower'
    tls_config:
        insecure_skip_verify: True
    metrics_path: /api/v2/metrics
    scrape_interval: 5s
    scheme: https
    bearer_token: <token_value>
    # basic_auth:
    #   username: admin
    #   password: password
    static_configs:
        - targets:
            - <tower_host>
```

For help configuring other aspects of Prometheus, such as alerts and service discovery configurations, refer to the Prometheus configuration docs.

If Prometheus is already running, you must restart it in order to apply the configuration changes by making a **POST** to the reload endpoint, or by killing the Prometheus process or service.

2. Use a browser to navigate to your graph in the Prometheus UI at `http://your_prometheus:9090/graph` and test out some queries. For example, you can query the current number of active Tower user sessions by executing: `awx_sessions_total{type="user"}`.



awx_sessions_total{type="user"}

Load time: 12ms
Resolution: 1s
Total time series: 1

✓ 🟥 awx_sessions_total{instance="awxweb:8013",job="awx",type="user"}

Refer to the metrics endpoint in the Tower API for your instance (`api/v2/metrics`) for more ways to query.

CHAPTER

THIRTEEN

SECRET HANDLING AND CONNECTION SECURITY

This document describes how Ansible Tower handles secrets and connections in a secure fashion.

13.1 Secret Handling

Ansible Tower manages three sets of secrets:

- user passwords for local Ansible Tower users
- secrets for Ansible Tower operational use (database password, message bus password, etc.)
- secrets for automation use (SSH keys, cloud credentials, external password vault credentials, etc.)

13.1.1 User passwords for local Ansible Tower users

Ansible Tower hashes local Ansible Tower user passwords with the PBKDF2 algorithm using a SHA256 hash. Users who authenticate via external account mechanisms (LDAP, SAML, OAuth, and others) do not have any password or secret stored.

13.1.2 Secret handling for Ansible Tower operational use

Ansible Tower contains the following secrets used operationally:

- /etc/tower/SECRET_KEY
  - A secret key used for encrypting automation secrets in the database (see below). If the SECRET_KEY changes or is unknown, no encrypted fields in the database will be accessible.
- /etc/tower/tower.{cert,key}
  - SSL certificate and key for the Ansible Tower web service. A self-signed cert/key is installed by default; the customer can provide a locally appropriate certificate and key.
- Database password in /etc/tower/conf.d/postgres.py and message bus password in /etc/tower/conf.d/channels.py
  - Passwords for connecting to Ansible Tower component services

These secrets are all stored unencrypted on the Ansible Tower server, as they are all needed to be read by the Ansible Tower service at startup in an automated fashion. All secrets are protected by Unix permissions, and restricted to root and the Ansible Tower service user awx.

If hiding of these secrets is required, the files that these secrets are read from are interpreted Python. These files can be adjusted to retrieve these secrets via some other mechanism anytime a service restarts. Doing so is a customer provided modification that may need to be reapplied every upgrade. Red Hat Support and Red Hat Consulting has examples of such modifications.

Note: If the secrets system is down, Tower will be unable to get the information and may fail in a way that would be recoverable once the service is restored. Using some redundancy on that system is highly recommended.

If, for any reason you believe the `SECRET_KEY` Tower generated for you has been compromised and needs to be regenerated, you can run a tool from the installer that behaves much like the Tower backup and restore tool. To generate a new secret key:

1. **Backup your Tower database before you do anything else!** Follow the procedure described in the Backing Up and Restoring Tower section of this guide.
2. Using the inventory from your install (same inventory with which you run backups/restores), run `setup.sh -k`.

A backup copy of the prior key is saved in `/etc/tower/`.

13.1.3 Secret handling for automation use

Ansible Tower stores a variety of secrets in the database that are either used for automation or are a result of automation. These secrets include:

- all secret fields of all credential types (passwords, secret keys, authentication tokens, secret cloud credentials)
- secret tokens and passwords for external services defined in Ansible Tower settings
- "password" type survey fields entries

To encrypt secret fields, Tower uses AES in CBC mode with a 256-bit key for encryption, PKCS7 padding, and HMAC using SHA256 for authentication. The encryption/decryption process derives the AES-256 bit encryption key from the SECRET_KEY (described above), the field name of the model field and the database assigned auto-incremented record ID. Thus, if any attribute used in the key generation process changes, Tower fails to correctly decrypt the secret. Ansible Tower is designed such that the SECRET_KEY is never readable in playbooks Ansible Tower launches, that these secrets are never readable by Tower users, and no secret field values are ever made available via the Ansible Tower REST API. If a secret value is used in a playbook, we recommend using *no_log* on the task so that it is not accidentally logged.

13.2 Connection Security

13.2.1 Internal Services

Ansible Tower connects to the following services as part of internal operation:

- PostgreSQL database
- A Redis key/value store

The connection to redis is over a local unix socket, restricted to the awx service user.

The connection to the PostgreSQL database is done via password authentication over TCP, either via localhost or remotely (external database). This connection can use PostgreSQL's built in support for SSL/TLS, as natively configured by the installer support. SSL/TLS protocols are configured by the default OpenSSL configuration.

13.2.2 External Access

Ansible Tower is accessed via standard HTTP/HTTPS on standard ports, provided by nginx. A self-signed cert/key is installed by default; the customer can provide a locally appropriate certificate and key. SSL/TLS algorithm support is configured in the /etc/nginx/nginx.conf configuration file. An "intermediate" profile is used by default, and can be configured by the customer. Customer changes must be reapplied on each update.

13.2.3 Managed Nodes

Ansible Tower also connects to managed machines and services as part of automation. All connections to managed machines are done via standard secure mechanism as specified such as SSH, WinRM, SSL/TLS, and so on - each of these inherits configuration from the system configuration for the feature in question (such as the system OpenSSL configuration).

CHAPTER

FOURTEEN

SECURITY BEST PRACTICES

Ansible Tower out-of-the-box is deployed in a secure fashion for use to automate typical environments. However, managing certain operating system environments, automation, and automation platforms, may require some additional best practices to ensure security. This document describes best practices for automation in a secure manner.

14.1 General best practices

An application is only as secure as the underlying system. To secure Red Hat Enterprise Linux, start with the release-appropriate security guide:

- For Red Hat Enterprise Linux 7: https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Security_Guide/
- For Red Hat Enterprise Linux 8: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/security_hardening/index

14.2 Understand the architecture of Ansible and Tower

Ansible and Ansible Tower comprise a general purpose, declarative, automation platform. That means that once an Ansible playbook is launched (via Tower, or directly on the command line), the playbook, inventory, and credentials provided to Ansible are considered to be the source of truth. If policies are desired around external verification of specific playbook content, job definition, or inventory contents, these processes must be undertaken before the automation is launched (whether via the Tower web UI, or the Tower API).

These can take many forms. The use of source control, branching, and mandatory code review is best practice for Ansible automation. There are many tools that can help create process flow around using source control in this manner.

At a higher level, many tools exist that allow for creation of approvals and policy-based actions around arbitrary workflows, including automation; these tools can then use Ansible via Tower's API to perform automation.

We recommend all customers of Ansible Tower select a secure default administrator password at time of installation. See Playbook setup for more information.

Ansible Tower exposes services on certain well-known ports, such as port 80 for HTTP traffic and port 443 for HTTPS traffic. We recommend that you do not expose Ansible Tower on the open internet, significantly reducing the threat surface of your installation.

14.3 Granting access

Granting access to certain parts of the system exposes security risks. Apply the following practices to help secure access:

14.3.1 Minimize administrative accounts

Minimizing the access to system administrative accounts is crucial for maintaining a secure system. A system administrator/root user can access, edit, and disrupt any system application. Keep the number of people/accounts with root access to as small of a group as possible. Do not give out *sudo* to *root* or *awx* (the Tower user) to untrusted users. Know that when restricting administrative access via mechanisms like *sudo*, that restricting to a certain set of commands may still give a wide range of access. Any command that allows for execution of a shell or arbitrary shell commands, or any command that can change files on the system, is fundamentally equivalent to full root access.

In a Tower context, any Tower 'system administrator' or 'superuser' account can edit, change, and update any inventory or automation definition in Tower. Restrict this to the minimum set of users possible for low-level Tower configuration and disaster recovery only.

14.3.2 Minimize local system access

Ansible Tower, when used with best practices, should not require local user access except for administrative purposes. Non-administrator users should not have access to the Tower system.

14.3.3 Remove access to credentials from users

If an automation credential is only stored in Tower, it can be further secured. Services such as OpenSSH can be configured to only allow credentials on connections from specific addresses. Credentials used by automation can be different than credentials used by system administrators for disaster-recovery or other ad-hoc management, allowing for easier auditing.

14.3.4 Enforce separation of duties

Different pieces of automation may need to access a system at different levels. For example, you may have low-level system automation that applies patches and performs security baseline checking, while a higher-level piece of automation deploys applications. By using different keys or credentials for each piece of automation, the effect of any one key vulnerability is minimized, while also allowing for easy baseline auditing.

14.4 Available resources

Several resources exist in Tower and elsewhere to ensure a secure platform. Consider utilizing the following functionality:

- *Audit and logging functionality*
- *Existing security functionality*
- *External account stores*
- *Django password policies*

14.4.1 Audit and logging functionality

For any administrative access, it is key to audit and watch for actions. For the system overall, this can be done via the built in audit support (https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Security_Guide/chap-system_auditing.html) and via the built-in logging support.

For Ansible Tower, this is done via the built-in Activity Stream support that logs all changes within Tower, as well as via the automation logs.

Best practices dictate collecting logging and auditing centrally, rather than reviewing it on the local system. It is recommended that Ansible Tower be configured to use whatever IDS and/or logging/auditing (Splunk) is standard in your environment. Ansible Tower includes built-in logging integrations for Elastic Stack, Splunk, Sumologic, Loggly, and more. See *Tower Logging and Aggregation* for more information.

14.4.2 Existing security functionality

Do not disable SELinux, and do not disable Tower's existing multi-tenant containment. Use Tower's role-based access control (RBAC) to delegate the minimum level of privileges required to run automation. Use Teams in Tower to assign permissions to groups of users rather than to users individually. See Role-Based Access Controls in the *Ansible Tower User Guide*.

14.4.3 External account stores

Maintaining a full set of users just in Tower can be a time-consuming task in a large organization, prone to error. Ansible Tower supports connecting to external account sources via *LDAP*, *SAML 2.0*, and certain *OAuth providers*. Using this eliminates a source of error when working with permissions.

14.4.4 Django password policies

Tower admins can leverage Django to set password policies at creation time via `AUTH_PASSWORD_VALIDATORS` to validate Tower user passwords. In the `custom.py` file located at `/etc/tower/conf.d` of your Tower instance, add the following code block example:

```
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.
↪UserAttributeSimilarityValidator',
```

(continues on next page)

(continued from previous page)

```
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
        'OPTIONS': {
            'min_length': 9,
        }
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
```

For more information, see Password management in Django in addition to the example posted above.

Be sure to restart your Tower instance for the change to take effect. See *Starting, Stopping, and Restarting Tower* for detail.

CHAPTER

FIFTEEN

RESOURCE PROFILING

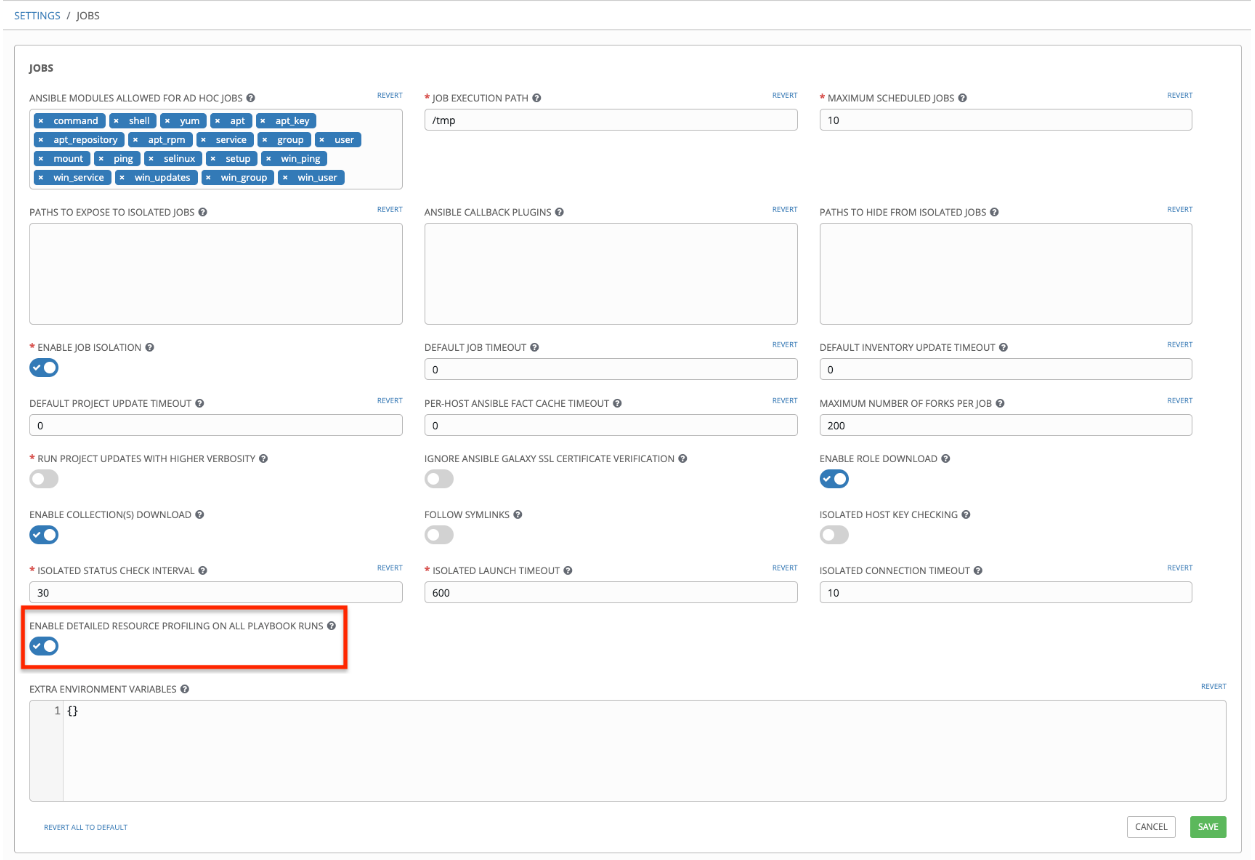
Ansible Tower has the ability to collect raw performance data including CPU, memory, and PID count during the execution of a playbook run. This is made possible by resource profiling capabilities provided by **Runner**, which uses Linux control groups (*'cgroups'*) to measure actual resource usage over time. For more information about cgroups, refer to Introduction to Control Groups.

When you install Ansible Tower, Tower automatically creates the cgroup so that Runner can use it.

15.1 Enable resource profiling

To enable Runner's resource profiling feature in the Tower User Interface:

1. From the left navigation bar, hover over the Settings (⚙) icon and select **Jobs** or click the **Jobs** tab from the Settings screen.
2. Use the toggle to turn on the **Enable Detailed Resource Profiling On All Playbook Runs** setting to collect data on all jobs.



SETTINGS / JOBS

JOBS

ANSIBLE MODULES ALLOWED FOR AD HOC JOBS: command, shell, yum, apt, apt_key, apt_repository, apt_rpm, service, group, user, mount, ping, selinux, setup, win_ping, win_service, win_updates, win_group, win_user

* JOB EXECUTION PATH: /tmp

* MAXIMUM SCHEDULED JOBS: 10

PATHS TO EXPOSE TO ISOLATED JOBS

ANSIBLE CALLBACK PLUGINS

PATHS TO HIDE FROM ISOLATED JOBS

* ENABLE JOB ISOLATION: on

DEFAULT JOB TIMEOUT: 0

DEFAULT INVENTORY UPDATE TIMEOUT: 0

DEFAULT PROJECT UPDATE TIMEOUT: 0

PER-HOST ANSIBLE FACT CACHE TIMEOUT: 0

MAXIMUM NUMBER OF FORKS PER JOB: 200

* RUN PROJECT UPDATES WITH HIGHER VERBOSITY: off

IGNORE ANSIBLE GALAXY SSL CERTIFICATE VERIFICATION: off

ENABLE ROLE DOWNLOAD: on

ENABLE COLLECTION(S) DOWNLOAD: on

FOLLOW SYMLINKS: off

ISOLATED HOST KEY CHECKING: off

* ISOLATED STATUS CHECK INTERVAL: 30

* ISOLATED LAUNCH TIMEOUT: 600

ISOLATED CONNECTION TIMEOUT: 10

ENABLE DETAILED RESOURCE PROFILING ON ALL PLAYBOOK RUNS: on

EXTRA ENVIRONMENT VARIABLES:

```
{}
```

3. Click **Save** to save your preferences.

After performance data has been collected for a job, it is stored under `/var/log/tower/playbook_profiling/<job_id>/`. **On a cluster**, performance data is stored on the Tower instance that executed the job. If a job is executed using an *isolated instance*, then the data is collected from the isolated node and stored on the controller that was used to deliver the job to the the isolated node.

Three data files (corresponding to CPU, memory, and PID count) are created for each task. Each file contains data in JSON text format; each line of the file will begin with a record separator (RS), continue with a JSON dictionary, and conclude with a line feed (LF) character. Note that if a task runs very quickly, it is possible that performance data may not be collected at all for that task, in which case one or more of the performance data files for that task will not be created.

Refer to the current Runner docs for more detail on what the performance data looks like, along with some actual sample data.

CHAPTER

SIXTEEN

THE *AWX-MANAGE* UTILITY

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

Warning: Running awx-manage commands via playbook is not recommended or supported.

16.1 Inventory Import

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

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

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

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

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

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

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

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

Note: Edits and additions to Inventory host variables persist beyond an inventory sync as long as --overwrite_vars is **not** set.

16.2 Cleanup of old data

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

- awx-manage cleanup_jobs [--help]

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

- awx-manage cleanup_activitystream [--help]

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

16.3 Cluster management

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

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

16.4 Token and session management

Ansible Tower supports the following commands for OAuth2 token management:

16.4.1 create_oauth2_token

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

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

New OAuth2 token for example_user: j89ia8OO79te6IAZ97L7E8bMgXCON2
```

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

16.4.2 revoke_oauth2_tokens

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

To revoke all existing OAuth2 tokens:

```
$ awx-manage revoke_oauth2_tokens
```

To revoke all OAuth2 tokens & their refresh tokens:

```
$ awx-manage revoke_oauth2_tokens --all
```

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

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

To revoke all OAuth2 tokens and refresh token for the user with the username `example_user` (specify actual username for `example_user` below):

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

16.4.3 cleartokens

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

16.4.4 expire_sessions

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

```
$ awx-manage expire_sessions
```

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

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

16.4.5 `clearsessions`

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

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

16.5 Analytics gathering

Use this command to gather analytics on-demand outside of the predefined window (default is 4 hours):

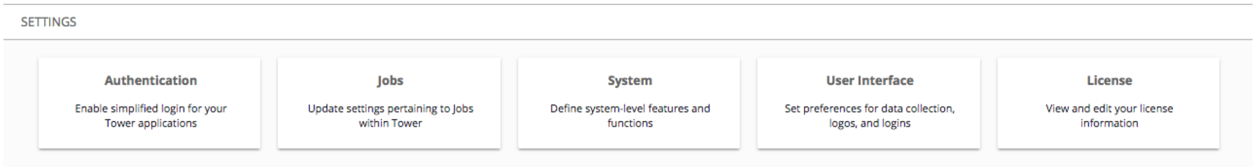
```
$ awx-manage gather_analytics --ship
```

CHAPTER

SEVENTEEN

TOWER CONFIGURATION

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



SETTINGS

Authentication
Enable simplified login for your Tower applications

Jobs
Update settings pertaining to Jobs within Tower

System
Define system-level features and functions

User Interface
Set preferences for data collection, logos, and logins

License
View and edit your license information

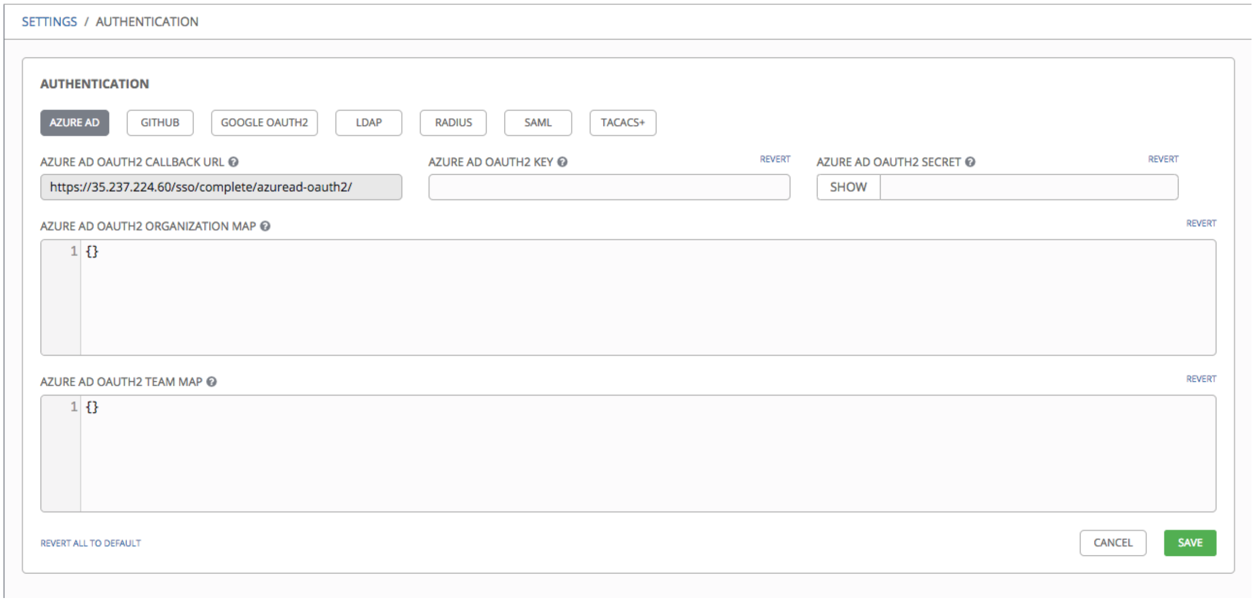
Each tab contains fields with a **Reset** button, allowing you to revert any value entered back to the default value. **Reset All** allows you to revert all the values in the Edit Tower Configuration to their factory default values.

Save applies changes you make, but it does not exit the edit dialog. To return to the Configure Tower screen, click the Settings (⚙) icon from the left navigation bar or use the breadcrumbs at the top of the current view.

17.1 Authentication

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

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



AZURE AD OAUTH2 CALLBACK URL: https://35.237.224.60/sso/complete/azuread-oauth2/

AZURE AD OAUTH2 KEY

AZURE AD OAUTH2 SECRET

AZURE AD OAUTH2 ORGANIZATION MAP

```
{}
```

AZURE AD OAUTH2 TEAM MAP

```
{}
```

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

Note: For more detail about each authentication type, refer to the *Setting up Social Authentication* section of the *Ansible Tower Administration Guide*.

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

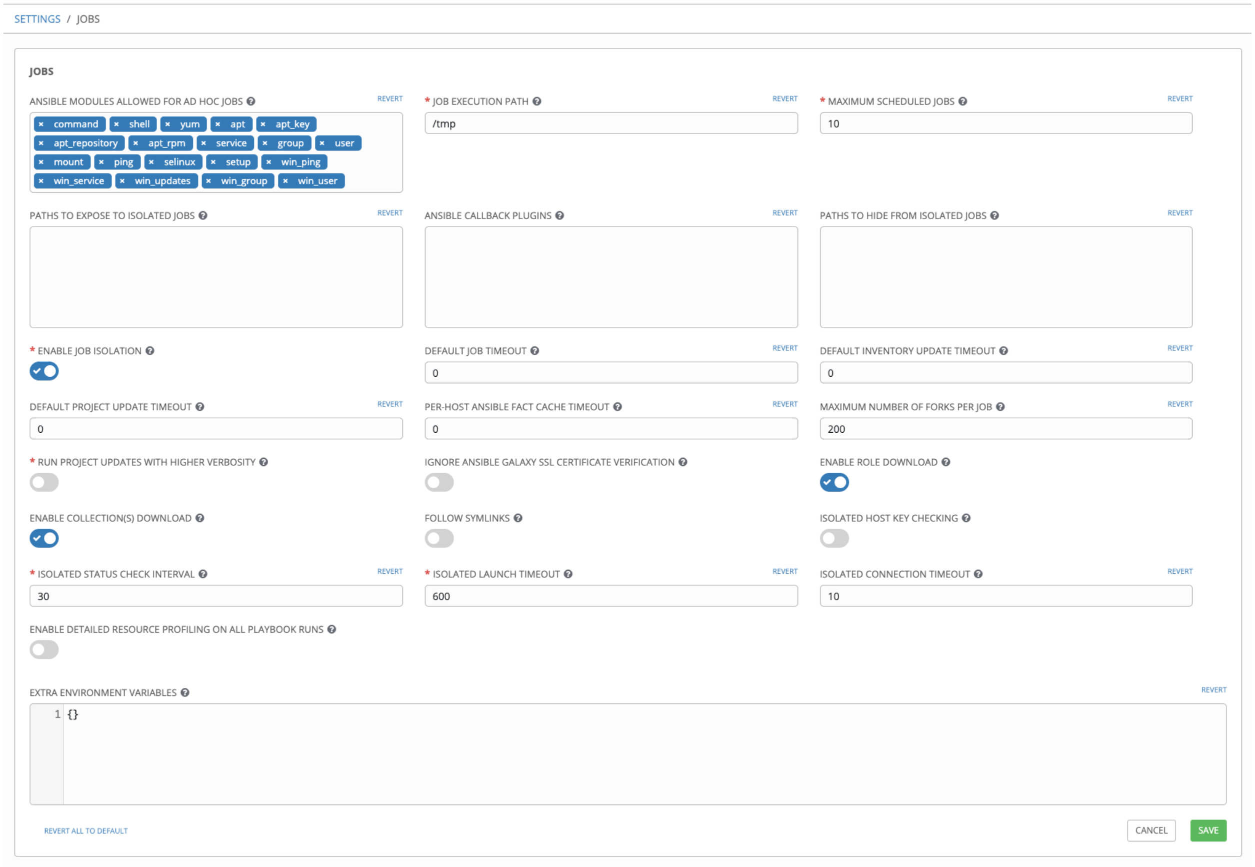
17.2 Jobs

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

1. From the left navigation bar, hover over the Settings (⚙) icon and select **Jobs** or click the **Jobs** tab from the Settings screen.

2. Set the configurable options from the fields provided. Click the tooltip ❓ icon next to the field that you need additional information or details about. Refer to the *Isolated Instance Groups* section for details about configuring durations for isolated instance groups. Refer to the Ansible Galaxy Support section for details about configuring Galaxy settings.

Note: The values for all the timeouts are in seconds.



SETTINGS / JOBS

JOBS

ANSIBLE MODULES ALLOWED FOR AD HOC JOBS: command, shell, yum, apt, apt_key, apt_repository, apt_rpm, service, group, user, mount, ping, selinux, setup, win_ping, win_service, win_updates, win_group, win_user

* JOB EXECUTION PATH: /tmp

* MAXIMUM SCHEDULED JOBS: 10

PATHS TO EXPOSE TO ISOLATED JOBS:

ANSIBLE CALLBACK PLUGINS:

PATHS TO HIDE FROM ISOLATED JOBS:

* ENABLE JOB ISOLATION: on

DEFAULT JOB TIMEOUT: 0

DEFAULT INVENTORY UPDATE TIMEOUT: 0

DEFAULT PROJECT UPDATE TIMEOUT: 0

PER-HOST ANSIBLE FACT CACHE TIMEOUT: 0

MAXIMUM NUMBER OF FORKS PER JOB: 200

* RUN PROJECT UPDATES WITH HIGHER VERBOSITY: off

IGNORE ANSIBLE GALAXY SSL CERTIFICATE VERIFICATION: off

ENABLE ROLE DOWNLOAD: on

ENABLE COLLECTION(S) DOWNLOAD: on

FOLLOW SYMLINKS: off

ISOLATED HOST KEY CHECKING: off

* ISOLATED STATUS CHECK INTERVAL: 30

* ISOLATED LAUNCH TIMEOUT: 600

ISOLATED CONNECTION TIMEOUT: 10

ENABLE DETAILED RESOURCE PROFILING ON ALL PLAYBOOK RUNS: off

EXTRA ENVIRONMENT VARIABLES:

```
{}
```

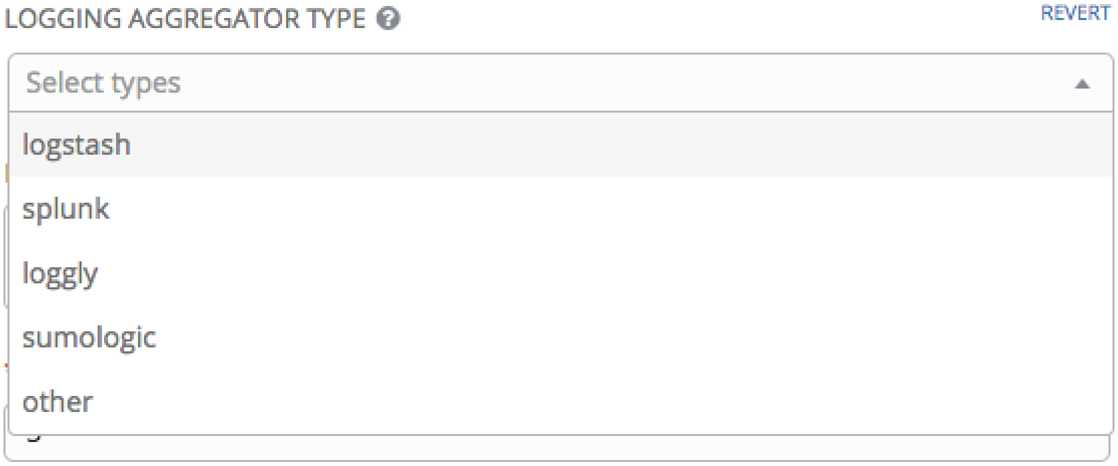
3. Click **Save** to apply the settings or **Cancel** to abandon the changes.

17.3 System

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

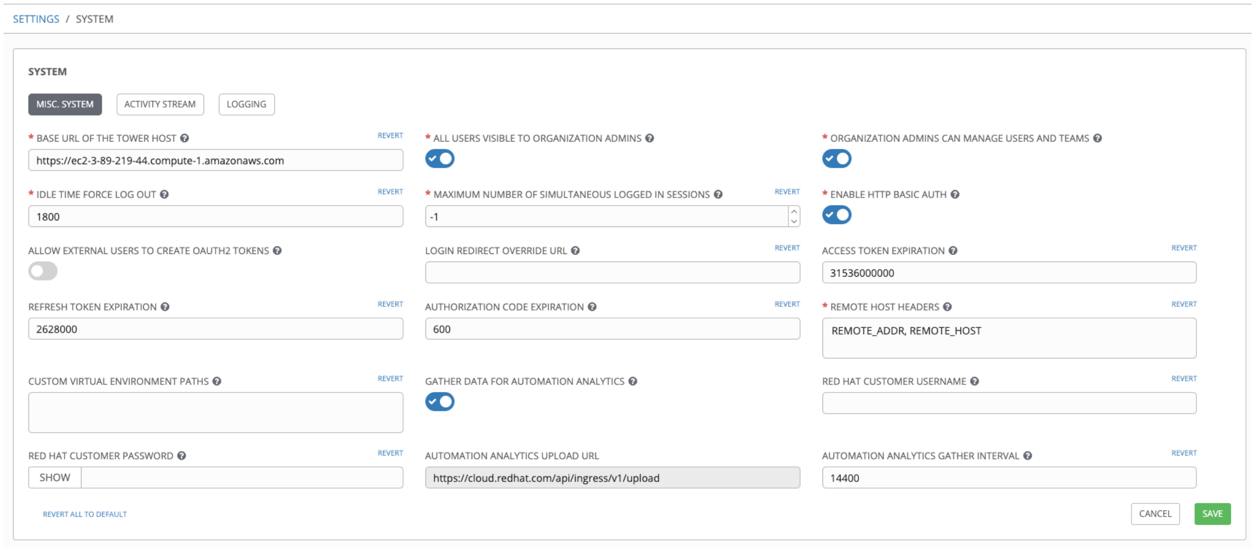
1. From the left navigation bar, hover over the Settings (⚙) icon and select **System** or click the **System** tab from the Settings screen.
2. The System window opens. Select an option from the row of tabs across the top of the window:

- **Misc. System**: define the base URL for the Tower host, enable tower administration alerts, and allow all users to be visible to organization administrators.
- **Activity Stream**: enable or disable activity stream.
- **Logging**: configure logging options based on the type you choose:



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

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



* BASE URL OF THE TOWER HOST: https://ec2-3-89-219-44.compute-1.amazonaws.com

* ALL USERS VISIBLE TO ORGANIZATION ADMINS: ✓

* ORGANIZATION ADMINS CAN MANAGE USERS AND TEAMS: ✓

* IDLE TIME FORCE LOG OUT: 1800

* MAXIMUM NUMBER OF SIMULTANEOUS LOGGED IN SESSIONS: -1

* ENABLE HTTP BASIC AUTH: ✓

ALLOW EXTERNAL USERS TO CREATE OAUTH2 TOKENS: ○

LOGIN REDIRECT OVERRIDE URL:

ACCESS TOKEN EXPIRATION: 31536000000

REFRESH TOKEN EXPIRATION: 2628000

AUTHORIZATION CODE EXPIRATION: 600

* REMOTE HOST HEADERS: REMOTE_ADDR, REMOTE_HOST

CUSTOM VIRTUAL ENVIRONMENT PATHS:

GATHER DATA FOR AUTOMATION ANALYTICS: ✓

RED HAT CUSTOMER USERNAME:

RED HAT CUSTOMER PASSWORD: SHOW

AUTOMATION ANALYTICS UPLOAD URL: https://cloud.redhat.com/api/ingress/v1/upload

AUTOMATION ANALYTICS GATHER INTERVAL: 14400

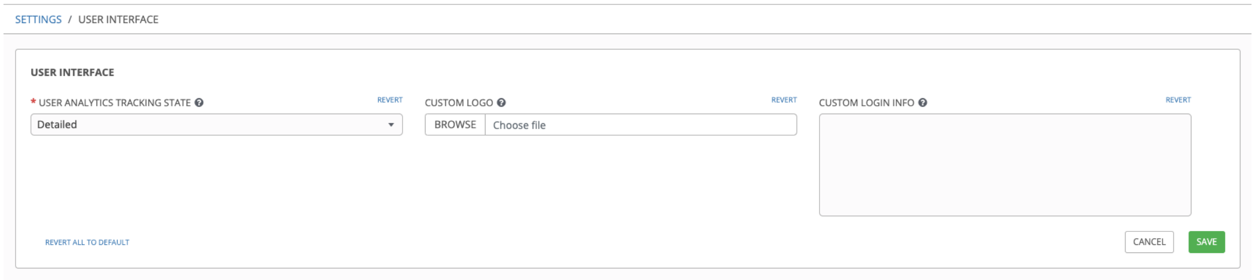
Note: The **Allow External Users to Create Oauth2 Tokens** setting is disabled by default. This ensures external users cannot *create* their own tokens. If you enable then disable it, any tokens created by external users in the meantime will still exist, and are not automatically revoked.

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

17.4 User Interface

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

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



USER ANALYTICS TRACKING STATE: Detailed

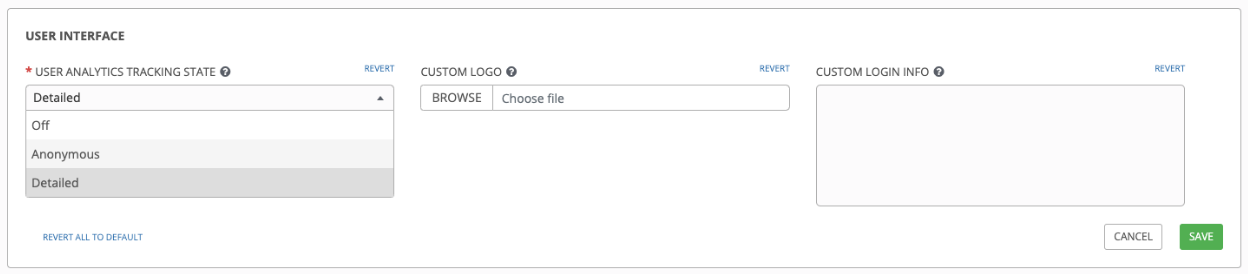
17.4.1 Usability Analytics and Data Collection

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

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

If you want to change how you participate in this analytics collection, you can opt out or change your settings using the Configure Tower user interface, accessible from the Settings ( ) icon from the left navigation bar.

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



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

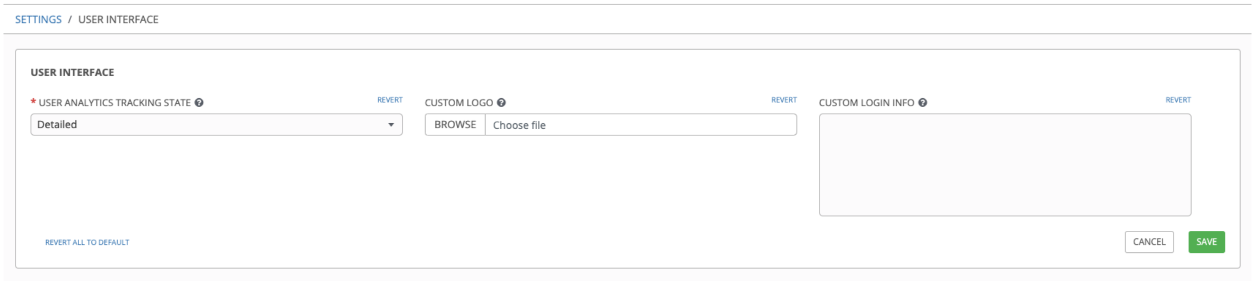
- **Off**: Prevents any data collection.
- **Anonymous**: Enables data collection without your specific user data.
- **Detailed**: Enables data collection including your specific user data.

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

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

17.4.2 Custom Logos and Images

Ansible Tower supports the use of a custom logo. You can add a custom logo by uploading an image; and supply a custom login message from the User Interface settings of the Settings (⚙) menu.



SETTINGS / USER INTERFACE

USER INTERFACE

* USER ANALYTICS TRACKING STATE: Detailed

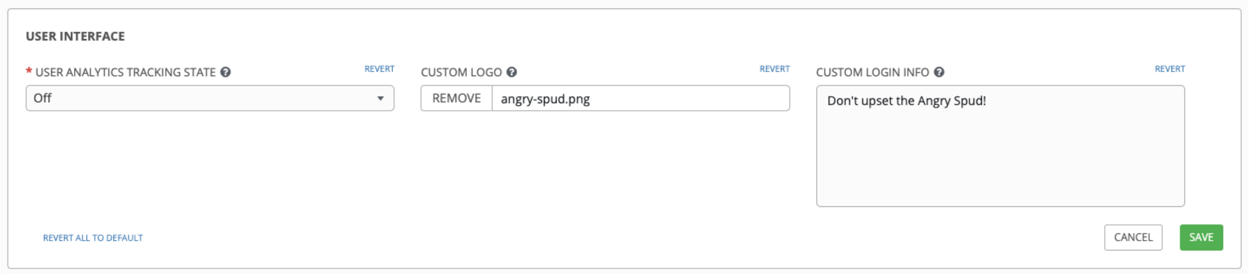
CUSTOM LOGO: Choose file

CUSTOM LOGIN INFO:

For the custom logo to look its best, use a `.png` file with a transparent background. GIF, PNG, and JPEG formats are supported.

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

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



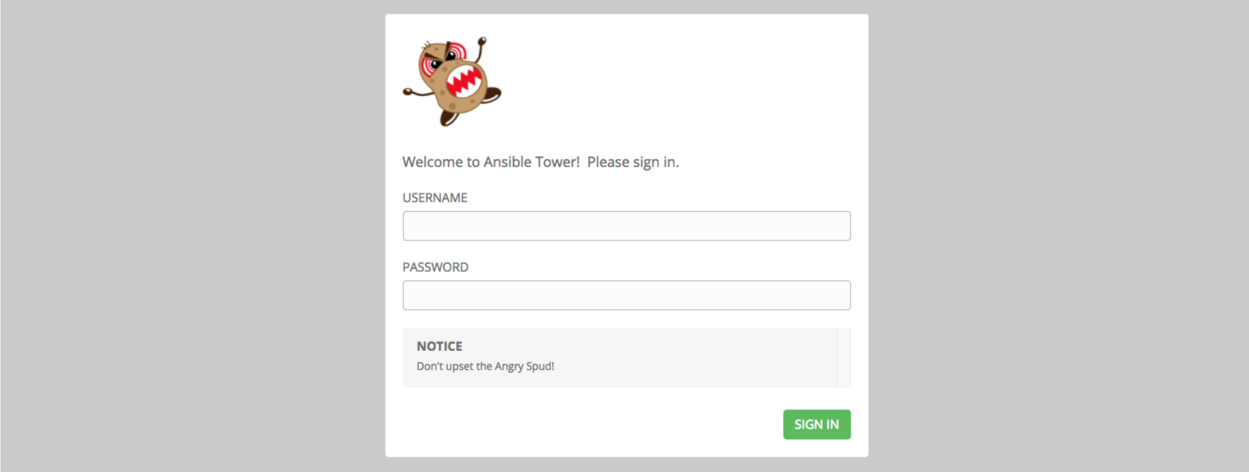
USER INTERFACE

* USER ANALYTICS TRACKING STATE: Off

CUSTOM LOGO: angry-spud.png

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

The Tower login dialog would look like this:



Welcome to Ansible Tower! Please sign in.

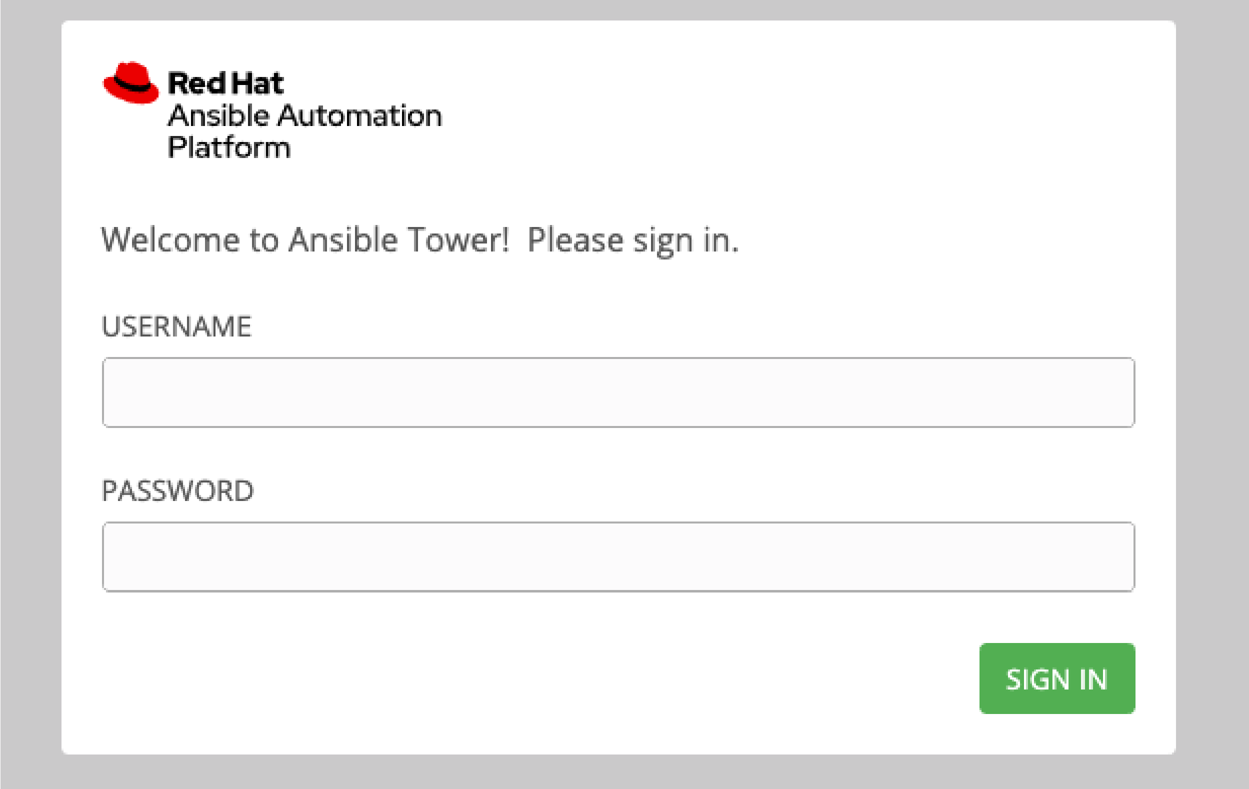
USERNAME

PASSWORD

NOTICE

Don't upset the Angry Spud!

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



Welcome to Ansible Tower! Please sign in.

USERNAME

PASSWORD

17.5 License

Starting with 3.8, Ansible Tower uses available subscriptions or a subscription manifest to authorize the use of Tower. Previously, Tower used a license key and a JSON dictionary of license metadata. **Even if you already have valid licenses from previous versions, you must still provide your credentials or a subscriptions manifest again upon upgrading to Tower 3.8.** To obtain your Tower subscription, you can either:

1. Provide your Red Hat or Satellite username and password on the license page.
2. Obtain a subscriptions manifest from your Subscription Allocations page on the customer portal. See Obtaining a subscriptions manifest for more detail.

If you **have** a Red Hat Ansible Automation Platform subscription, use your Red Hat customer credentials when you launch Tower to access your subscription information (see instructions below).

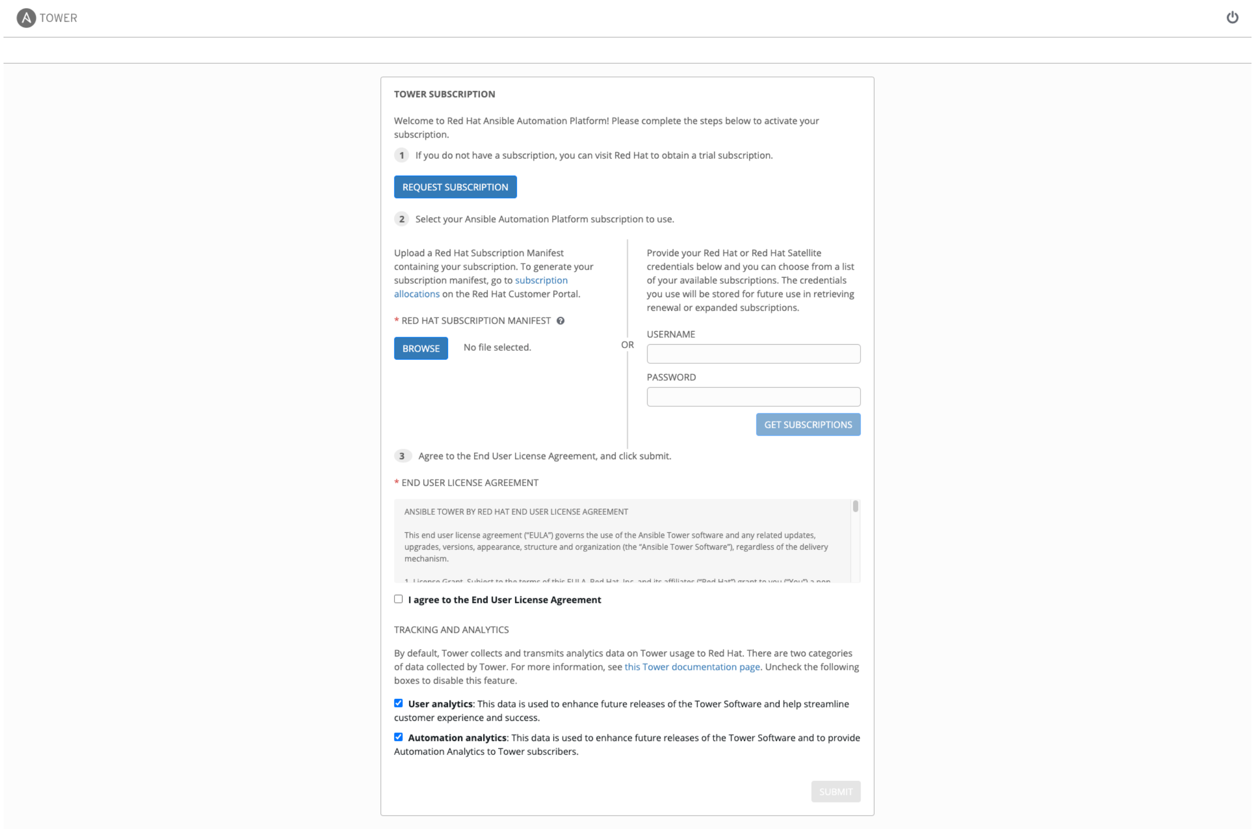
If you **do not** have a Red Hat Ansible Automation Platform subscription, you can request a trial subscription here or click **Request Subscription** and follow the instructions to request one.

Disconnected environments with Satellite will be able to use the login flow on vm-based installations if they have configured subscription manager on the Tower instance to connect to their Satellite instance. Recommended workarounds for disconnected environments **without Satellite** include [1] downloading a manifest from access.redhat.com in a connected environment, then uploading it to the disconnected Tower instance, or [2] connecting to the Internet through a proxy server.

Note: In order to use a disconnected environment, it is necessary to have a valid Ansible Tower entitlement attached to your Satellite organization's manifest. This can be confirmed by using `hammer subscription list \--organization <org_name>`.

If you have issues with the subscription you have received, please contact your Sales Account Manager or Red Hat Customer Service at https://access.redhat.com/support/contact/customerService/.

When Tower launches for the first time, the Tower Subscription screen automatically displays.



TOWER SUBSCRIPTION

Welcome to Red Hat Ansible Automation Platform! Please complete the steps below to activate your subscription.

1 If you do not have a subscription, you can visit Red Hat to obtain a trial subscription.

2 Select your Ansible Automation Platform subscription to use.

Upload a Red Hat Subscription Manifest containing your subscription. To generate your subscription manifest, go to subscription allocations on the Red Hat Customer Portal.

* RED HAT SUBSCRIPTION MANIFEST

No file selected.

Provide your Red Hat or Red Hat Satellite credentials below and you can choose from a list of your available subscriptions. The credentials you use will be stored for future use in retrieving renewal or expanded subscriptions.

USERNAME

PASSWORD

3 Agree to the End User License Agreement, and click submit.

* END USER LICENSE AGREEMENT

ANSIBLE TOWER BY RED HAT END USER LICENSE AGREEMENT

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

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

☐ **I agree to the End User License Agreement**

TRACKING AND ANALYTICS

By default, Tower collects and transmits analytics data on Tower usage to Red Hat. There are two categories of data collected by Tower. For more information, see this Tower documentation page. Uncheck the following boxes to disable this feature.

☑ **User analytics**: This data is used to enhance future releases of the Tower Software and help streamline customer experience and success.

☑ **Automation analytics**: This data is used to enhance future releases of the Tower Software and to provide Automation Analytics to Tower subscribers.

Use your Red Hat credentials (username and password) to retrieve and import your subscription, or upload a subscription manifest you generate from https://access.redhat.com/management/subscription_allocations.

1. Enter your Red Hat customer credentials (username and password) and click **Get Subscriptions**. Use your Satellite username/password if your Tower cluster nodes are registered to Satellite via Subscription Manager. See Installing Satellite instances on Tower for more information.

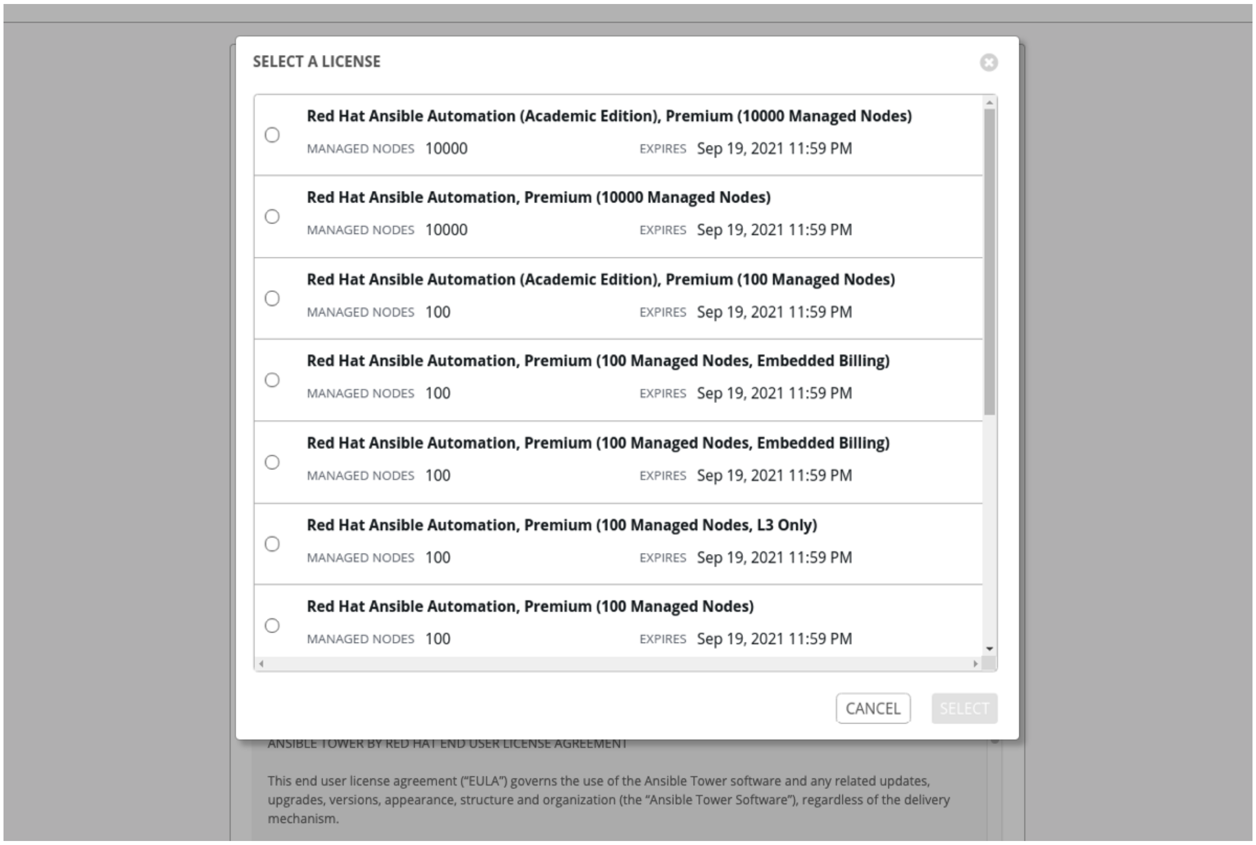
Alternatively, if you have a subscriptions manifest, you can upload it by browsing to the location where the file is saved to upload it (the subscription manifest is the complete .zip file, not its component parts). See Obtaining a subscriptions manifest for more detail.

Note: If the **Browse** button is grayed-out, clear the username and password fields to enable the **Browse** button.

2. The subscription metadata is then retrieved from the RHSM/Satellite API, or from the manifest provided.

- If it is a subscription manifest, Tower will use the first valid subscription included in your manifest file. This is why it is important to only include the subscription you want applied to the Tower installation.
- If you entered your credential information (username/password), Tower retrieves your configured subscription service. Then it prompts you to choose the subscription you want to run (the example below shows multiple subscriptions) and entitles Tower with that metadata. You can log in over time and retrieve new subscriptions if you have renewed.

Note: When your subscription expires (you can check this on the License settings in the Configure Tower screen of the UI), you will need to renew it in Tower by one of these two methods.



SELECT A LICENSE

○ Red Hat Ansible Automation (Academic Edition), Premium (10000 Managed Nodes)
MANAGED NODES 10000 EXPIRES Sep 19, 2021 11:59 PM

○ Red Hat Ansible Automation, Premium (10000 Managed Nodes)
MANAGED NODES 10000 EXPIRES Sep 19, 2021 11:59 PM

○ Red Hat Ansible Automation (Academic Edition), Premium (100 Managed Nodes)
MANAGED NODES 100 EXPIRES Sep 19, 2021 11:59 PM

○ Red Hat Ansible Automation, Premium (100 Managed Nodes, Embedded Billing)
MANAGED NODES 100 EXPIRES Sep 19, 2021 11:59 PM

○ Red Hat Ansible Automation, Premium (100 Managed Nodes, Embedded Billing)
MANAGED NODES 100 EXPIRES Sep 19, 2021 11:59 PM

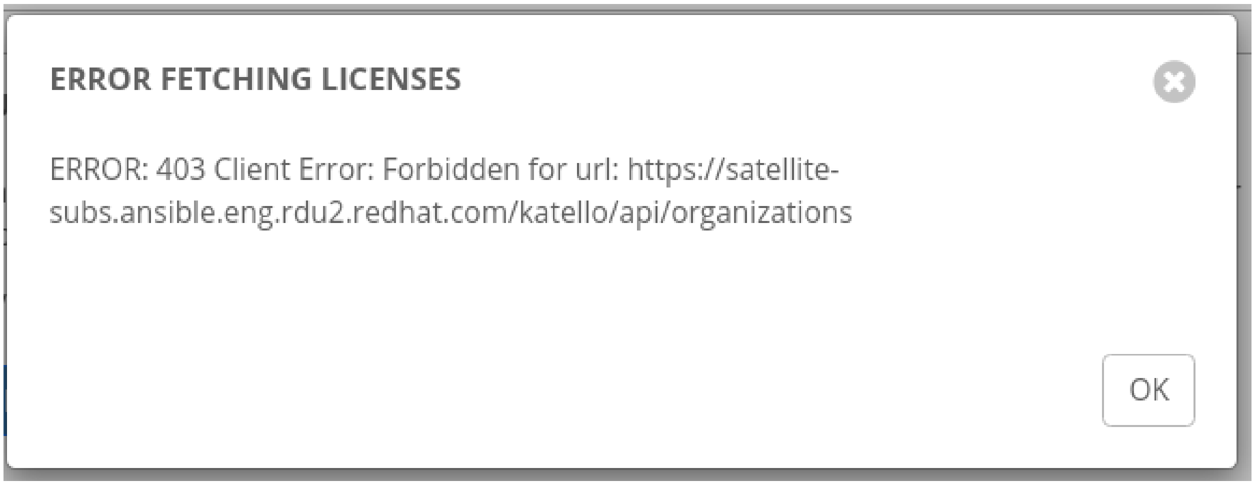
○ Red Hat Ansible Automation, Premium (100 Managed Nodes, L3 Only)
MANAGED NODES 100 EXPIRES Sep 19, 2021 11:59 PM

○ Red Hat Ansible Automation, Premium (100 Managed Nodes)
MANAGED NODES 100 EXPIRES Sep 19, 2021 11:59 PM

ANSIBLE TOWER BY RED HAT END USER LICENSE AGREEMENT

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

If you encounter the following error message, you will need the proper permissions required for the Satellite user with which the Tower admin uses to apply a subscription.



ERROR FETCHING LICENSES

ERROR: 403 Client Error: Forbidden for url: https://satellite-subs.ansible.eng.rdu2.redhat.com/katello/api/organizations

The Satellite username/password is used to query the Satellite API for existing subscriptions. From the Satellite API, Tower gets back some metadata about those subscriptions, then filter through to find valid subscriptions that you could apply, which are then displayed as valid subscription options in the UI.

The following Satellite roles grant proper access:

- Custom with `view_subscriptions` and `view_organizations` filter

- Viewer
- Administrator
- Organization Admin
- Manager

As the *Custom* role is the most restrictive of these, this is the recommend role to use for your Tower integration. Refer to the Satellite documentation on managing users and roles for more detail.

Note: The System Administrator role is not equivalent to the Administrator user checkbox, and will not provide sufficient permissions to access the subscriptions API page.

---

3. Proceed by checking the **End User License Agreement**.
4. The bottom half of the license screen involves analytics data collection. This helps Red Hat improve the product by delivering you a much better user experience. For more information about data collection, refer to *Usability Analytics and Data Collection*. This option is checked by default, but you may opt out of any of the following:

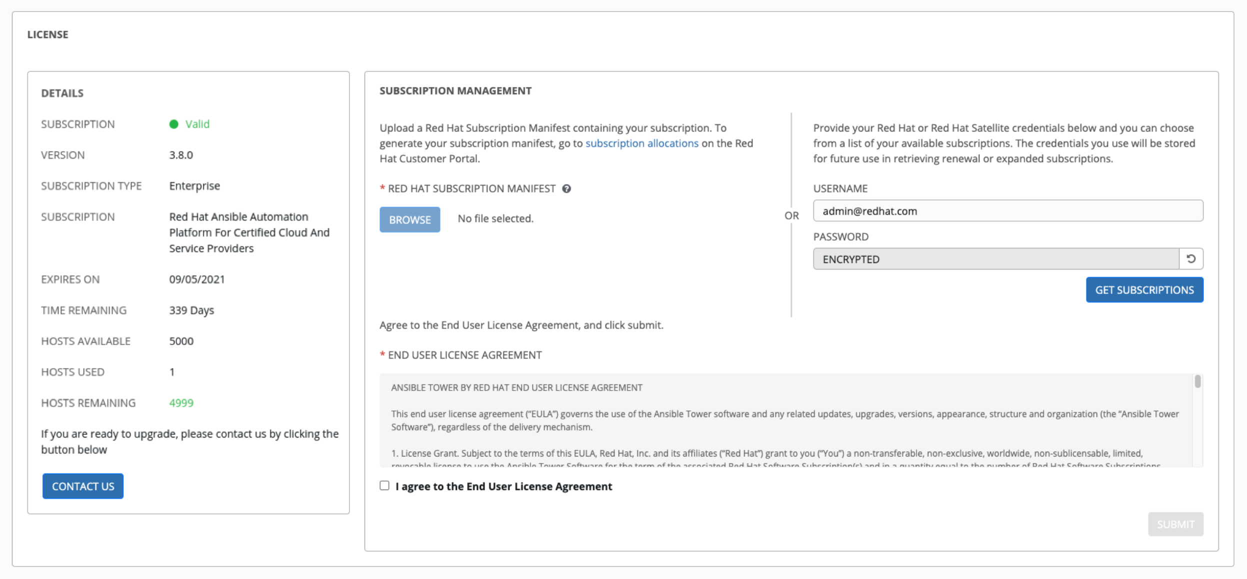
- **User analytics** collects data from the Tower User Interface.
- **Automation analytics** provides a high level analysis of your automation with Ansible Tower, which is used to help you identify trends and anomalous use of Tower. For opt-in of Automation Analytics to have any effect, your instance of Ansible Tower **must** be running on Red Hat Enterprise Linux. See instructions described in the *Automation Analytics* section.

Note: At this time, Automation Insights is not supported when Ansible Tower is running in the OpenShift Container Platform. You may change your analytics data collection preferences at any time, as described in the *Usability Analytics and Data Collection* section.

---

5. After you have specified your tracking and analytics preferences, click **Submit**.

Once your subscription has been accepted, Tower briefly displays the license screen and navigates you to the Dashboard of the Ansible Tower interface. For later reference, you can return to the license screen by clicking the Settings (⚙) icon from the left navigation bar and select the **License** tab from the Settings screen.



LICENSE

DETAILS

SUBSCRIPTION	● Valid
VERSION	3.8.0
SUBSCRIPTION TYPE	Enterprise
SUBSCRIPTION	Red Hat Ansible Automation Platform For Certified Cloud And Service Providers
EXPIRES ON	09/05/2021
TIME REMAINING	339 Days
HOSTS AVAILABLE	5000
HOSTS USED	1
HOSTS REMAINING	4999

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

SUBSCRIPTION MANAGEMENT

Upload a Red Hat Subscription Manifest containing your subscription. To generate your subscription manifest, go to subscription allocations on the Red Hat Customer Portal.

* RED HAT SUBSCRIPTION MANIFEST

No file selected.

OR

Provide your Red Hat or Red Hat Satellite credentials below and you can choose from a list of your available subscriptions. The credentials you use will be stored for future use in retrieving renewal or expanded subscriptions.

USERNAME

admin@redhat.com

PASSWORD

ENCRYPTED

Agree to the End User License Agreement, and click submit.

* END USER LICENSE AGREEMENT

ANSIBLE TOWER BY RED HAT END USER LICENSE AGREEMENT

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

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

☐ **I agree to the End User License Agreement**

CHAPTER

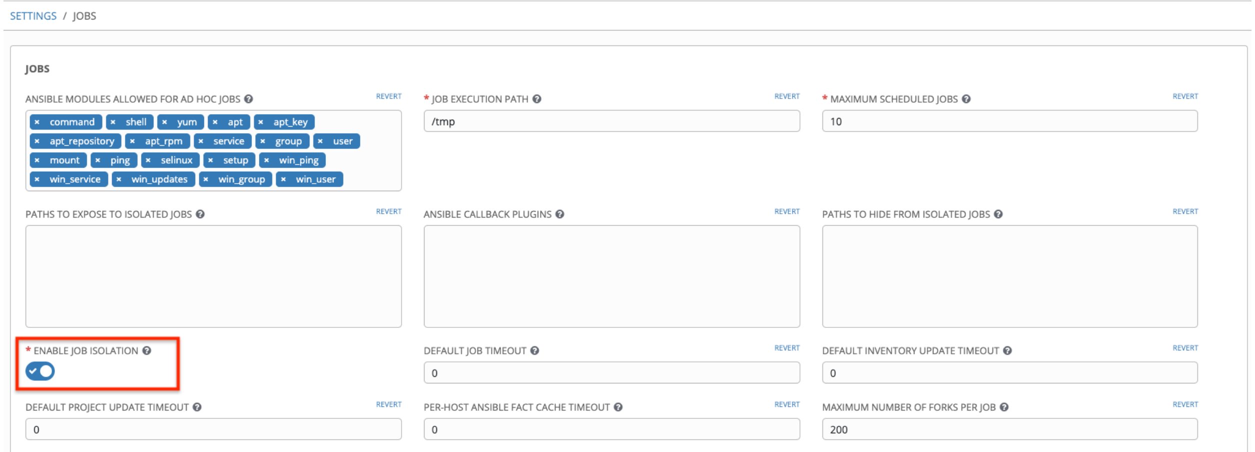
EIGHTEEN

BUBBLEWRAP FUNCTIONALITY AND VARIABLES

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

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

1. Click the Settings (⚙) icon from the left navigation bar.
2. Click the **Jobs** tab.
3. In the **Enable Job Isolation**, change the toggle button selection to **OFF** to disable bubblewrap support or select **ON** to enable it.



SETTINGS / JOBS

JOBS

ANSIBLE MODULES ALLOWED FOR AD HOC JOBS: command, shell, yum, apt, apt_key, apt_repository, apt_rpm, service, group, user, mount, ping, selinux, setup, win_ping, win_service, win_updates, win_group, win_user

* JOB EXECUTION PATH: /tmp

* MAXIMUM SCHEDULED JOBS: 10

PATHS TO EXPOSE TO ISOLATED JOBS:

ANSIBLE CALLBACK PLUGINS:

PATHS TO HIDE FROM ISOLATED JOBS:

* ENABLE JOB ISOLATION: ON

DEFAULT JOB TIMEOUT: 0

DEFAULT INVENTORY UPDATE TIMEOUT: 0

DEFAULT PROJECT UPDATE TIMEOUT: 0

PER-HOST ANSIBLE FACT CACHE TIMEOUT: 0

MAXIMUM NUMBER OF FORKS PER JOB: 200

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

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

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

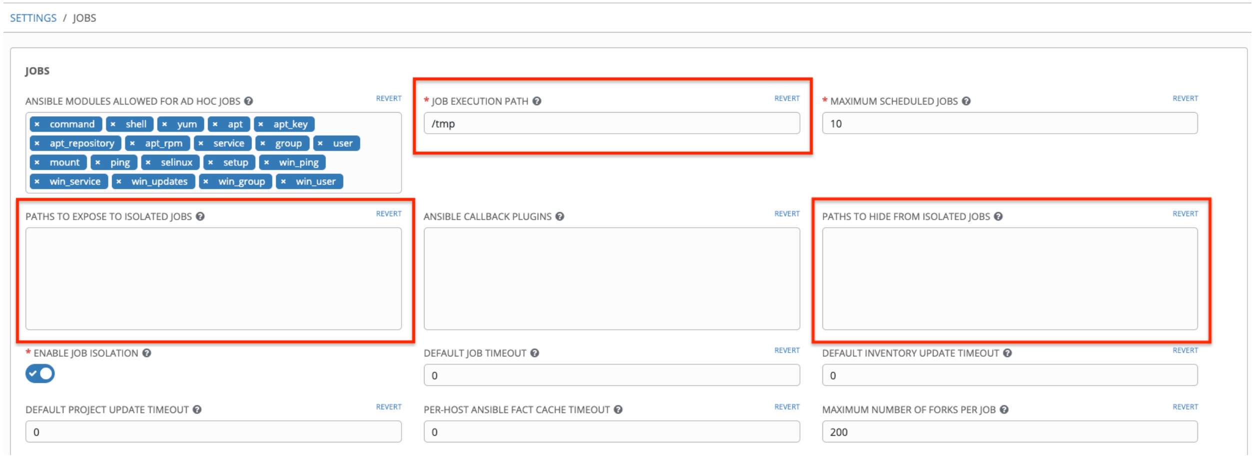
```
AWX_PROOT_HIDE_PATHS = ['/list/of/', '/paths']
```

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

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

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

The above fields can be found in the Jobs Settings window:



SETTINGS / JOBS

JOBS

ANSIBLE MODULES ALLOWED FOR AD HOC JOBS: command, shell, yum, apt, apt_key, apt_repository, apt_rpm, service, group, user, mount, ping, selinux, setup, win_ping, win_service, win_updates, win_group, win_user

* JOB EXECUTION PATH: /tmp

* MAXIMUM SCHEDULED JOBS: 10

PATHS TO EXPOSE TO ISOLATED JOBS

ANSIBLE CALLBACK PLUGINS

PATHS TO HIDE FROM ISOLATED JOBS

* ENABLE JOB ISOLATION

DEFAULT JOB TIMEOUT: 0

DEFAULT INVENTORY UPDATE TIMEOUT: 0

DEFAULT PROJECT UPDATE TIMEOUT: 0

PER-HOST ANSIBLE FACT CACHE TIMEOUT: 0

MAXIMUM NUMBER OF FORKS PER JOB: 200

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

CHAPTER

NINETEEN

TOKEN-BASED AUTHENTICATION

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

For details on using the `manage` utility to create tokens, refer to the *Token and session management* section.

19.1 Managing OAuth 2 Applications and Tokens

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

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

```
    {
    "id": 1,
    "type": "o_auth2_application",
    "url": "/api/v2/applications/2/",
    "related": {
        "tokens": "/api/v2/applications/2/tokens/"
    },
    "summary_fields": {
        "organization": {
            "id": 1,
            "name": "Default",
            "description": ""
        },
        "user_capabilities": {
            "edit": true,
            "delete": true
        },
        "tokens": {
            "count": 0,
            "results": []
        }
    },
```

(continues on next page)

(continued from previous page)

```
    "created": "2018-07-02T21:16:45.824400Z",
    "modified": "2018-07-02T21:16:45.824514Z",
    "name": "My Application",
    "description": "",
    "client_id": "Ecmc6RjjhKUOWJzDYEP8TZ35P3dvsKt0AKdIjgHV",
    "client_secret":
↪"7Ft7ym8MpE54yWGUNvxxg6KqGwPFsyhYn9QQfYHlgBxai74Qp1GE4zsvJduOfSFkTfWFnPzYpxqcRsy1KacD0HH0vOAQUDJDC
↪",
    "client_type": "confidential",
    "redirect_uris": "",
    "authorization_grant_type": "password",
    "skip_authorization": false,
    "organization": 1
}
```

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

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

19.1.1 Access Rules for Applications

Access rules for applications are as follows:

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

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

- POST to the `/api/v2/tokens/` endpoint with `application` and `scope` fields to point to the related application and specify token scope
- POST to the `/api/v2/applications/<pk>/tokens/` endpoint with the `scope` field (the parent application will be automatically linked)

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

```
{
    "id": 4,
    "type": "o_auth2_access_token",
    "url": "/api/v2/tokens/4/",
    "related": {
        "user": "/api/v2/users/1/",
        "application": "/api/v2/applications/1/",
        "activity_stream": "/api/v2/tokens/4/activity_stream/"
},
    "summary_fields": {
        "application": {
            "id": 1,
```

(continues on next page)

(continued from previous page)

```
            "name": "Default application for root",
            "client_id": "mcU5J5uGQcEQMgAZyr5JUnM3BqBJpgbgL9fLOVch"
        },
        "user": {
            "id": 1,
            "username": "root",
            "first_name": "",
            "last_name": ""
        }
    },
    "created": "2018-02-23T14:39:32.618932Z",
    "modified": "2018-02-23T14:39:32.643626Z",
    "description": "App Token Test",
    "user": 1,
    "token": "************",
    "refresh_token": "************",
    "application": 1,
    "expires": "2018-02-24T00:39:32.618279Z",
    "scope": "read"
},
```

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

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

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

19.1.2 Access rules for tokens

Access rules for tokens are as follows:

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

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

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

The easiest and most common way to obtain an OAuth 2 token is to create a personal access token at the `/api/v2/users/<userid>/personal_tokens/` endpoint, as shown in this example below:

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

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

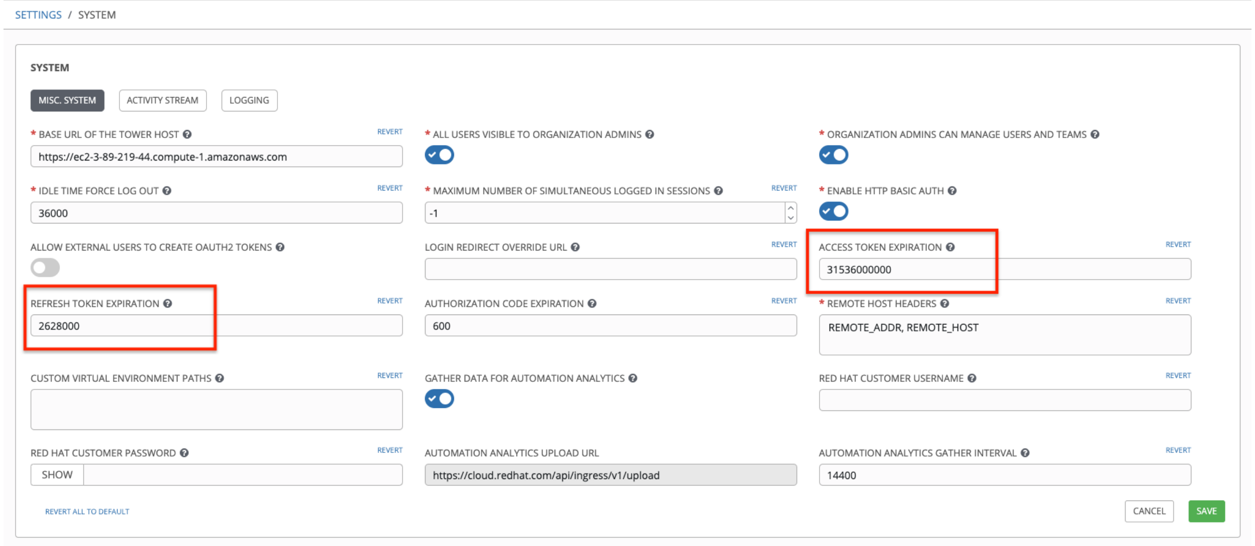
Following is an example of using the personal token to access an API endpoint using curl:

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

In Ansible Tower, the OAuth 2 system is built on top of the Django Oauth Toolkit, which provides dedicated endpoints for authorizing, revoking, and refreshing tokens. These endpoints can be found under the `/api/v2/users/<USER_ID>/personal_tokens/` endpoint, which also provides detailed examples on some typical usage of those endpoints. These special OAuth 2 endpoints only support using the `x-www-form-urlencoded` **Content-type**, so none of the `api/o/*` endpoints accept `application/json`.

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

Alternatively, you can add tokens for users through the Tower User Interface, as well as configure the expiration of an access token and its associated refresh token (if applicable).



SETTINGS / SYSTEM

SYSTEM

* BASE URL OF THE TOWER HOST: https://ec2-3-89-219-44.compute-1.amazonaws.com

* ALL USERS VISIBLE TO ORGANIZATION ADMINS: ✓ (on)

* ORGANIZATION ADMINS CAN MANAGE USERS AND TEAMS: ✓ (on)

* IDLE TIME FORCE LOG OUT: 36000

* MAXIMUM NUMBER OF SIMULTANEOUS LOGGED IN SESSIONS: -1

* ENABLE HTTP BASIC AUTH: ✓ (on)

ALLOW EXTERNAL USERS TO CREATE OAUTH2 TOKENS: ○ (off)

LOGIN REDIRECT OVERRIDE URL:

ACCESS TOKEN EXPIRATION: 31536000000

REFRESH TOKEN EXPIRATION: 2628000

AUTHORIZATION CODE EXPIRATION: 600

* REMOTE HOST HEADERS: REMOTE_ADDR, REMOTE_HOST

CUSTOM VIRTUAL ENVIRONMENT PATHS:

GATHER DATA FOR AUTOMATION ANALYTICS: ✓ (on)

RED HAT CUSTOMER USERNAME:

RED HAT CUSTOMER PASSWORD: SHOW

AUTOMATION ANALYTICS UPLOAD URL: https://cloud.redhat.com/api/ingress/v1/upload

AUTOMATION ANALYTICS GATHER INTERVAL: 14400

19.2.1 Token scope mask over RBAC system

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

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

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

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

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

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

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

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

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

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

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

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

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

Similarly, using a token:

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

19.3 Application Functions

This page lists OAuth 2 utility endpoints used for authorization, token refresh, and revoke. The `/api/o/` endpoints are not meant to be used in browsers and do not support HTTP GET. The endpoints prescribed here strictly follow RFC specifications for OAuth 2, so use that for detailed reference. The following is an example of the typical usage of these endpoints in Tower, in particular, when creating an application using various grant types:

- Authorization Code
- Password

Note: You can perform any of the application functions described here using the Tower User Interface. Refer to the Applications section of the *Ansible Tower User Guide* for more detail.

19.3.1 Application using `authorization code` grant type

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

Note:

> You can only use the `authorization code` type to acquire an access token when using an application. When integrating an external webapp with Ansible Tower, that webapp may need to create OAuth2 Tokens on behalf of users in that other webapp. Creating an application in Tower with the `authorization code` grant type is the preferred way to do this because:
>
> - this allows an external application to obtain a token from Tower for a user, using their credentials.
> - compartmentalized tokens issued for a particular application allows those tokens to be easily managed (revoke all tokens associated with that application without having to revoke *all* tokens in the system, for example)

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

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


.. _`Django-oauth-toolkit simple test application`: http://django-oauth-toolkit.
↪herokuapp.com/consumer/
```

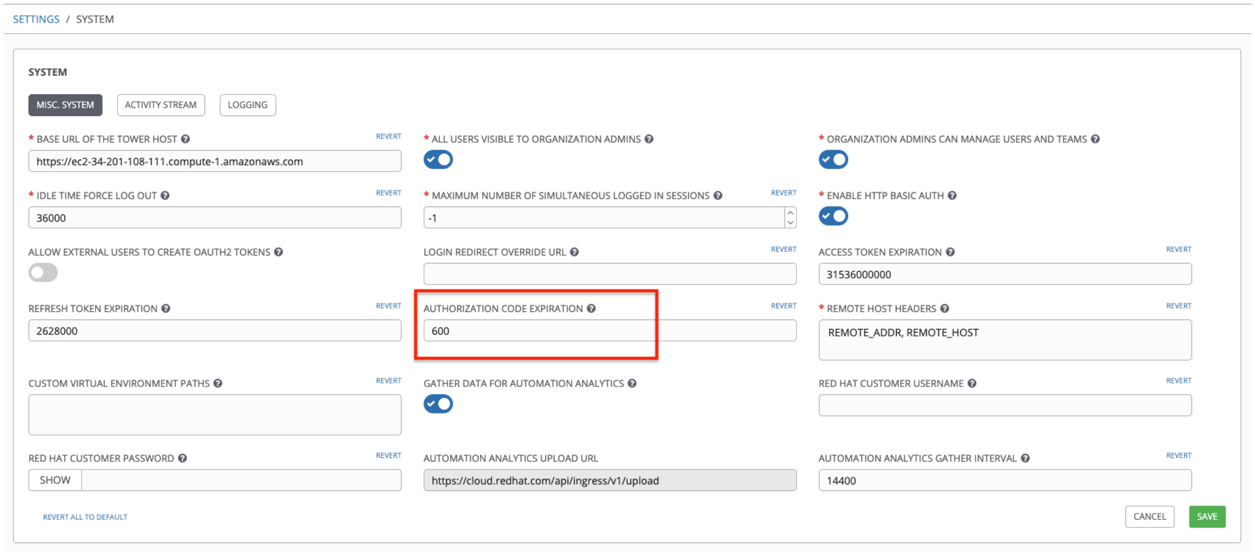
The workflow that occurs when you issue a **GET** to the `authorize` endpoint from the client application with the `response_type`, `client_id`, `redirect_uris`, and `scope`:

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

3. Tower responds with the `access_token`, `token_type`, `refresh_token`, and `expires_in`.

Refer to Django's Test Your Authorization Server toolkit to test this flow.

You may specify the number of seconds an authorization code remains valid in the Configure Tower - **System** settings:



SETTINGS / SYSTEM

SYSTEM

* BASE URL OF THE TOWER HOST: https://ec2-34-201-108-111.compute-1.amazonaws.com

* ALL USERS VISIBLE TO ORGANIZATION ADMINS: ✓

* ORGANIZATION ADMINS CAN MANAGE USERS AND TEAMS: ✓

* IDLE TIME FORCE LOG OUT: 36000

* MAXIMUM NUMBER OF SIMULTANEOUS LOGGED IN SESSIONS: -1

* ENABLE HTTP BASIC AUTH: ✓

ALLOW EXTERNAL USERS TO CREATE OAUTH2 TOKENS: ○

LOGIN REDIRECT OVERRIDE URL:

ACCESS TOKEN EXPIRATION: 31536000000

REFRESH TOKEN EXPIRATION: 2628000

AUTHORIZATION CODE EXPIRATION: 600

* REMOTE HOST HEADERS: REMOTE_ADDR, REMOTE_HOST

CUSTOM VIRTUAL ENVIRONMENT PATHS:

GATHER DATA FOR AUTOMATION ANALYTICS: ✓

RED HAT CUSTOMER USERNAME:

RED HAT CUSTOMER PASSWORD: SHOW

AUTOMATION ANALYTICS UPLOAD URL: https://cloud.redhat.com/api/ingress/v1/upload

AUTOMATION ANALYTICS GATHER INTERVAL: 14400

Requesting an access token after this duration will fail. The duration defaults to 600 seconds (10 minutes), based on the RFC6749 recommendation.

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

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

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

19.3.2 Application using `password` grant type

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

```
{
    "id": 6,
    "type": "application",
    ...
    "name": "Default Application",
    "user": 1,
    "client_id": "gwSPoasWSdNkMDtBN3Hu2WYQpPWCO9SwUEsKK22l",
    "client_secret":
↪"fI6ZpfocHYBGfm1tP92r0yIgCyfRdDQt0Tos9L8a4fNsJjQQMwp9569eIaUBsaVDgt2eiwOGe0bg5m5vCSstClZmtdy359RVx
↪",
```

(continues on next page)

(continued from previous page)

```
    "client_type": "confidential",
    "redirect_uris": "",
    "authorization_grant_type": "password",
    "skip_authorization": false
}
```

Logging in is not required for `password` grant type, so you can simply use curl to acquire a personal access token through the `/api/v2/tokens/` endpoint:

```
curl -k --user <user>:<password> -H "Content-type: application/json" \
-X POST \
--data '{
    "description": "Token for Nagios Monitoring app",
    "application": 1,
    "scope": "write"
}' \
https://<tower>/api/v2/tokens/
```

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

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

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

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

19.4 Application Token Functions

This section describes the refresh and revoke functions associated with tokens. Everything that follows (Refreshing and revoking tokens at the `/api/o/` endpoints) can currently only be done with application tokens.

19.4.1 Refresh an existing access token

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

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

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

```
curl -X POST \
    -d "grant_type=refresh_token&refresh_token=AL0NK9TTpv0qp54dGbC4VUZtsZ9r8z" \
    -u
↪"gwSPoasWSdNkMDtBN3Hu2WYQpPWCO9SwUEsKK22l:fI6ZpfocHYBGfm1tP92r0yIgCyfRdDQt0Tos9L8a4fNsJjQQMwp9569e
↪" \
    http://<tower>/api/o/token/ -i
```

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

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

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

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

{"access_token": "NDInWxGJI4iZgqpsreujjbvzCfJqgR", "token_type": "Bearer", "expires_in
↪": 315360000000, "refresh_token": "DqOrmz8bx3srlHkZNKmDpqA86bnQkT", "scope": "read␣
↪write"}
```

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

19.4.2 Revoke an access token

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

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

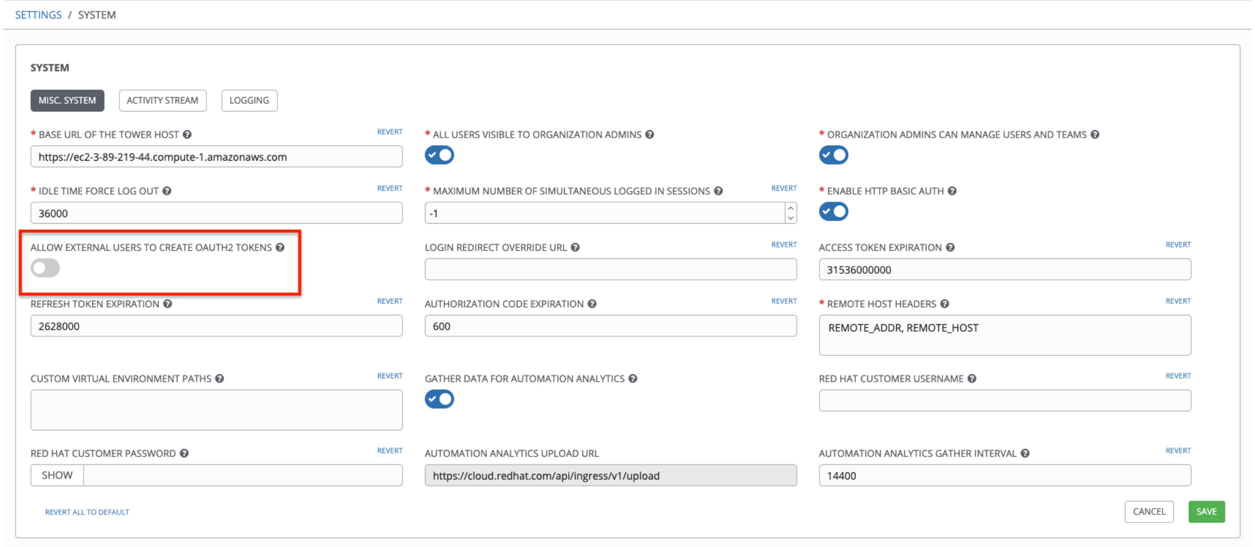
```
curl -X POST -d "token=rQONsve372fQwuc2pn76k3IHDCYpi7" \
-u
↪"gwSPoasWSdNkMDtBN3Hu2WYQpPWCO9SwUEsKK22l:fI6ZpfocHYBGfm1tP92r0yIgCyfRdDQt0Tos9L8a4fNsJjQQMwp9569e
↪" \
http://<tower>/api/o/revoke_token/ -i
```

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

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

Alternatively, you can use the `manage` utility, *revoke_oauth2_tokens*, to revoke tokens as described in the the *Token and session management* section.

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



* BASE URL OF THE TOWER HOST: https://ec2-3-89-219-44.compute-1.amazonaws.com

* ALL USERS VISIBLE TO ORGANIZATION ADMINS: ●

* ORGANIZATION ADMINS CAN MANAGE USERS AND TEAMS: ●

* IDLE TIME FORCE LOG OUT: 36000

* MAXIMUM NUMBER OF SIMULTANEOUS LOGGED IN SESSIONS: -1

* ENABLE HTTP BASIC AUTH: ●

ALLOW EXTERNAL USERS TO CREATE OAUTH2 TOKENS: ○

LOGIN REDIRECT OVERRIDE URL:

ACCESS TOKEN EXPIRATION: 31536000000

REFRESH TOKEN EXPIRATION: 2628000

AUTHORIZATION CODE EXPIRATION: 600

* REMOTE HOST HEADERS: REMOTE_ADDR, REMOTE_HOST

CUSTOM VIRTUAL ENVIRONMENT PATHS:

GATHER DATA FOR AUTOMATION ANALYTICS: ●

RED HAT CUSTOMER USERNAME:

RED HAT CUSTOMER PASSWORD: SHOW

AUTOMATION ANALYTICS UPLOAD URL: https://cloud.redhat.com/api/ingress/v1/upload

AUTOMATION ANALYTICS GATHER INTERVAL: 14400

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

CHAPTER

TWENTY

SETTING UP SOCIAL AUTHENTICATION

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

Account authentication can be configured in the Ansible Tower User Interface and saved to the PostgreSQL database. For instructions, refer to the *Tower Configuration* section.

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

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

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

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

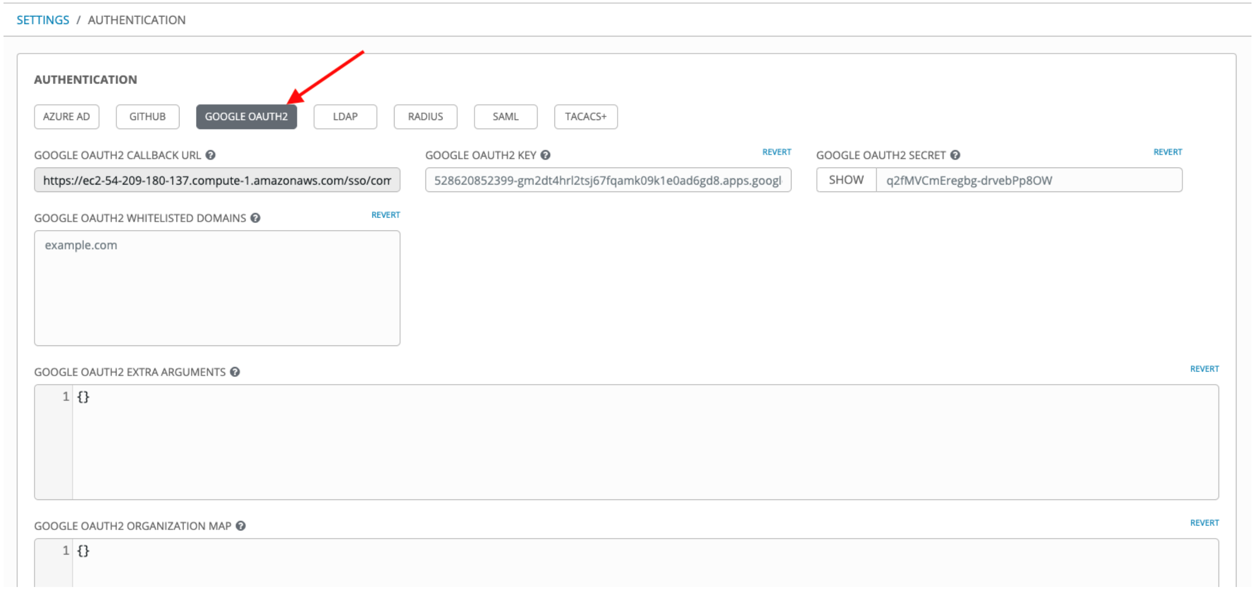
20.1 Google OAuth2 Settings

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

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

The Azure AD tab displays initially by default.

2. Select the **Google OAuth2** tab.



GOOGLE OAUTH2 CALLBACK URL: https://ec2-54-209-180-137.compute-1.amazonaws.com/sso/com

GOOGLE OAUTH2 KEY: 528620852399-gm2dt4hrl2tsj67fqamk09k1e0ad6gd8.apps.googl

GOOGLE OAUTH2 SECRET: q2fMVCmEregbg-drvebPp8OW

GOOGLE OAUTH2 WHITELISTED DOMAINS: example.com

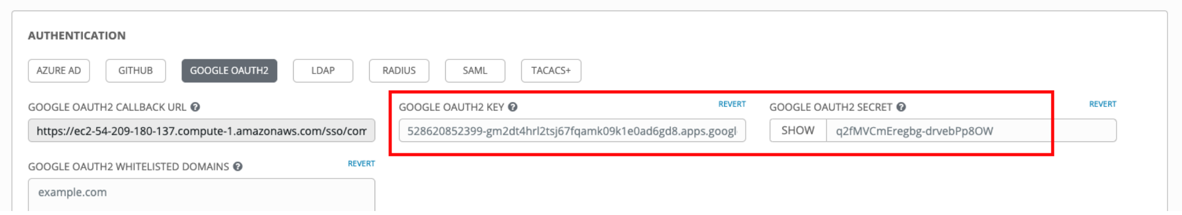
GOOGLE OAUTH2 EXTRA ARGUMENTS: {}

GOOGLE OAUTH2 ORGANIZATION MAP: {}

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

3. The following fields are also pre-populated. If not, use the credentials Google supplied during the web application setup process, and look for the values with the same format as the ones shown in the example below:

- Copy and paste Google's Client ID into the **Google OAuth2 Key** field.
- Copy and paste Google's Client secret into the **Google OAuth2 Secret** field.



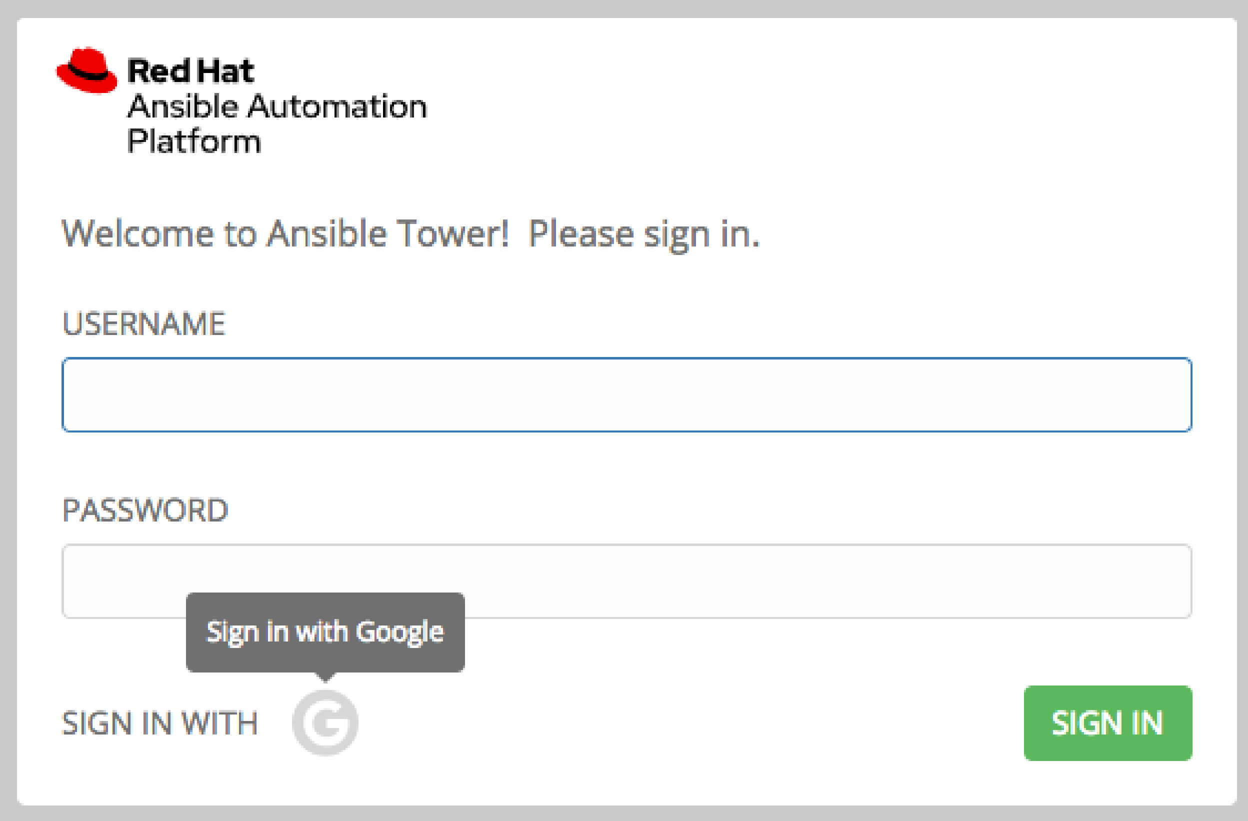
GOOGLE OAUTH2 CALLBACK URL: https://ec2-54-209-180-137.compute-1.amazonaws.com/sso/com

GOOGLE OAUTH2 KEY: 528620852399-gm2dt4hrl2tsj67fqamk09k1e0ad6gd8.apps.googl

GOOGLE OAUTH2 SECRET: q2fMVCmEregbg-drvebPp8OW

GOOGLE OAUTH2 WHITELISTED DOMAINS: example.com

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



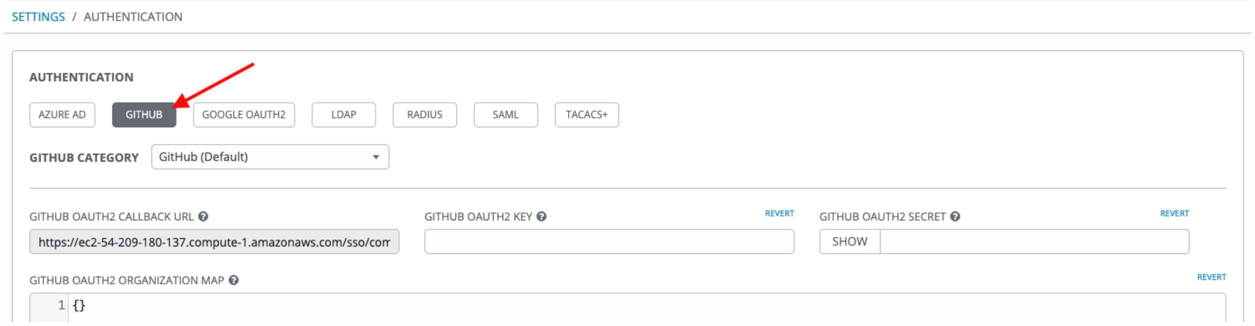
20.2 GitHub OAuth2 Settings

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

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

The Azure AD tab displays initially by default.

2. Select the **GitHub** tab.



GITHUB CATEGORY: GitHub (Default)

GITHUB OAUTH2 CALLBACK URL: https://ec2-54-209-180-137.compute-1.amazonaws.com/sso/com

GITHUB OAUTH2 KEY

GITHUB OAUTH2 SECRET

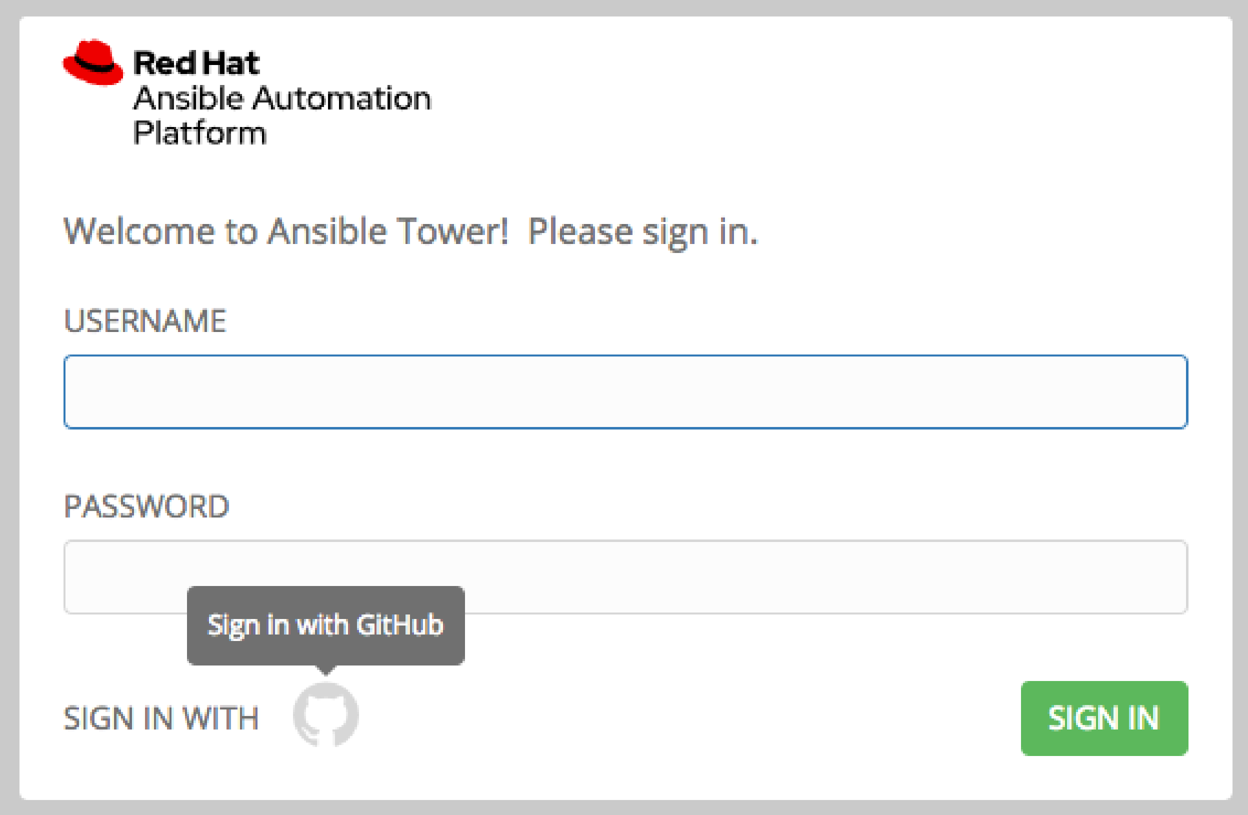
GITHUB OAUTH2 ORGANIZATION MAP

```
1 {}
```

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

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

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



20.2.1 GitHub Org Settings

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

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

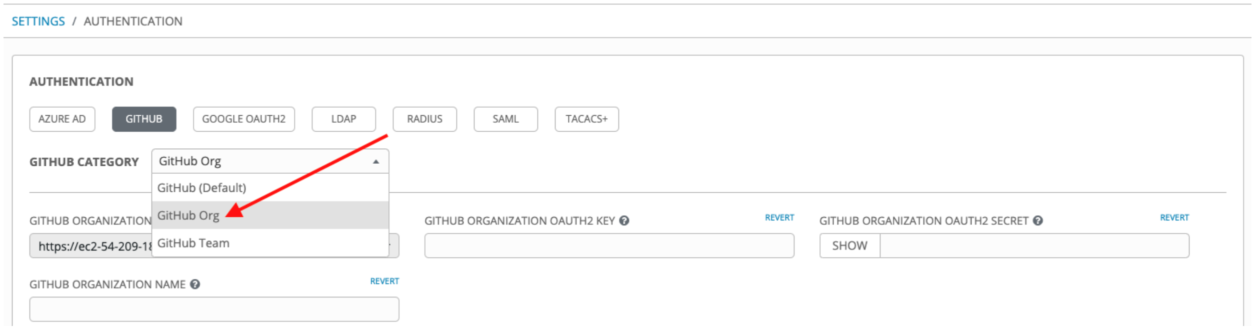
You can limit users who can login to Tower by limiting only those in an organization or on a team within an organization.

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

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

The Azure AD tab displays initially by default.

2. Select the **GitHub** tab and select **GutHub Org** from the GitHub Category drop-down menu list.



GITHUB CATEGORY: GitHub Org

GitHub (Default)
GitHub Org
GitHub Team

GITHUB ORGANIZATION: https://ec2-54-209-1

GITHUB ORGANIZATION OAUTH2 KEY

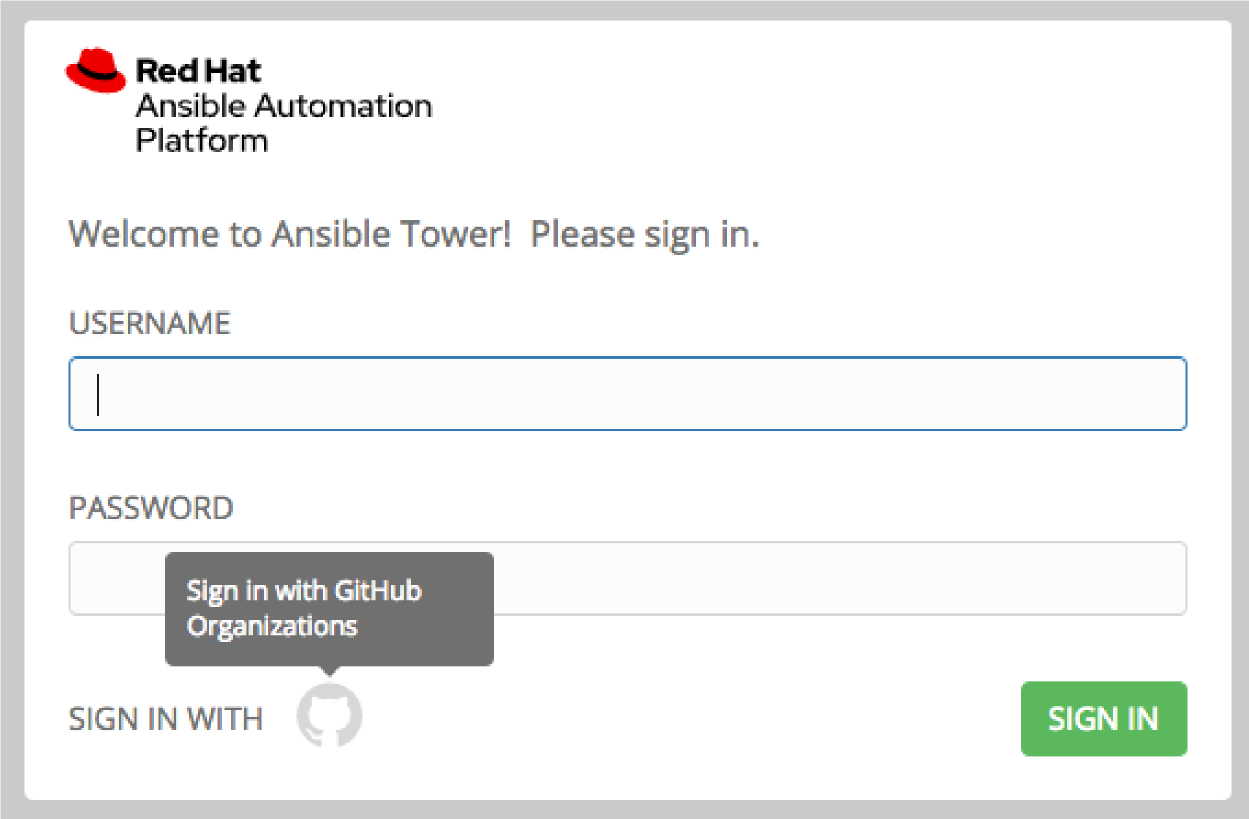
GITHUB ORGANIZATION OAUTH2 SECRET

GITHUB ORGANIZATION NAME

3. The **GitHub Organization OAuth2 Callback URL** field is already pre-populated and non-editable.

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

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



Welcome to Ansible Tower! Please sign in.

20.2.2 GitHub Team Settings

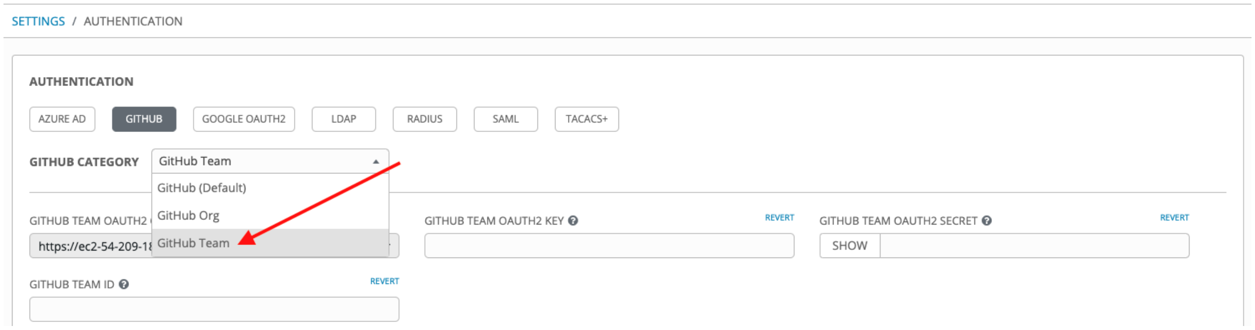
To set up social authentication for a GitHub Team, you will need to obtain an OAuth2 key and secret for a web application. To do this, you must first register your team-owned application at `https://github.com/organizations/<yourorg>/settings/applications`. In order to register the application, you must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

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

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

The Azure AD tab displays initially by default.

3. Select the **GitHub** tab and select **GutHub Team** from the GitHub Category drop-down menu list.



GITHUB CATEGORY: GitHub Team

GITHUB TEAM OAUTH2: https://ec2-54-209-1

4. The **GitHub Team OAuth2 Callback URL** field is already pre-populated and non-editable.

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

5. Copy and paste GitHub's Client ID into the **GitHub Team OAuth2 Key** field.

6. Copy and paste GitHub's Client Secret into the **GitHub Team OAuth2 Secret** field.

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

8. Click **Save** when done.

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

20.3 Organization and Team Mapping

20.3.1 Organization mapping

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

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

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

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

- If **None**, organization admins will not be updated.
- If **True**, all users using account authentication will automatically be added as admins of the organization.
- If **False**, no account authentication users will be automatically added as admins of the organization.
- If a string or list of strings, specifies the usernames and emails for users who will be added to the organization. Strings beginning and ending with `/` will be compiled into regular expressions; modifiers `i` (case-insensitive) and `m` (multi-line) may be specified after the ending `/`.

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

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

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

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

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

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

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

20.3.2 Team mapping

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

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

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

- If **None**, team members will not be updated.
- If **True/False**, all social auth users will be added/removed as team members.

- If a string or list of strings, specifies expressions used to match users. User will be added as a team member if the username or email matches. Strings beginning and ending with / will be compiled into regular expressions; modifiers i (case-insensitive) and m (multi-line) may be specified after the ending /.

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

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

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

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

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

```
SOCIAL_AUTH_USER_FIELDS = []
```

CHAPTER TWENTYONE

SETTING UP ENTERPRISE AUTHENTICATION

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

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

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

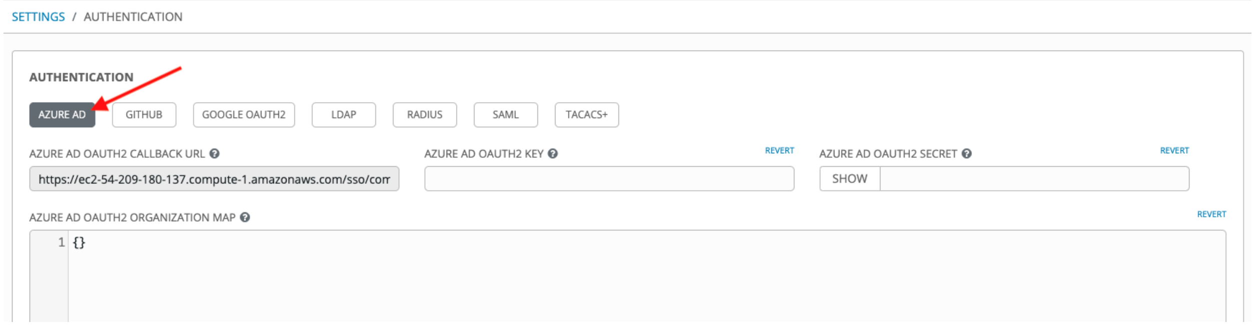
- Enterprise users can only be created via the first successful login attempt from remote authentication backend.
- Enterprise users cannot be created/authenticated if non-enterprise users with the same name has already been created in Tower.
- Tower passwords of enterprise users should always be empty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

21.1 Azure Active Directory (AD)

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

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

2. Select the **Azure AD** tab if it is not already the default view.



SETTINGS / AUTHENTICATION

AUTHENTICATION

AZURE AD OAUTH2 CALLBACK URL: https://ec2-54-209-180-137.compute-1.amazonaws.com/sso/com

AZURE AD OAUTH2 KEY

AZURE AD OAUTH2 SECRET

AZURE AD OAUTH2 ORGANIZATION MAP

```
{}
```

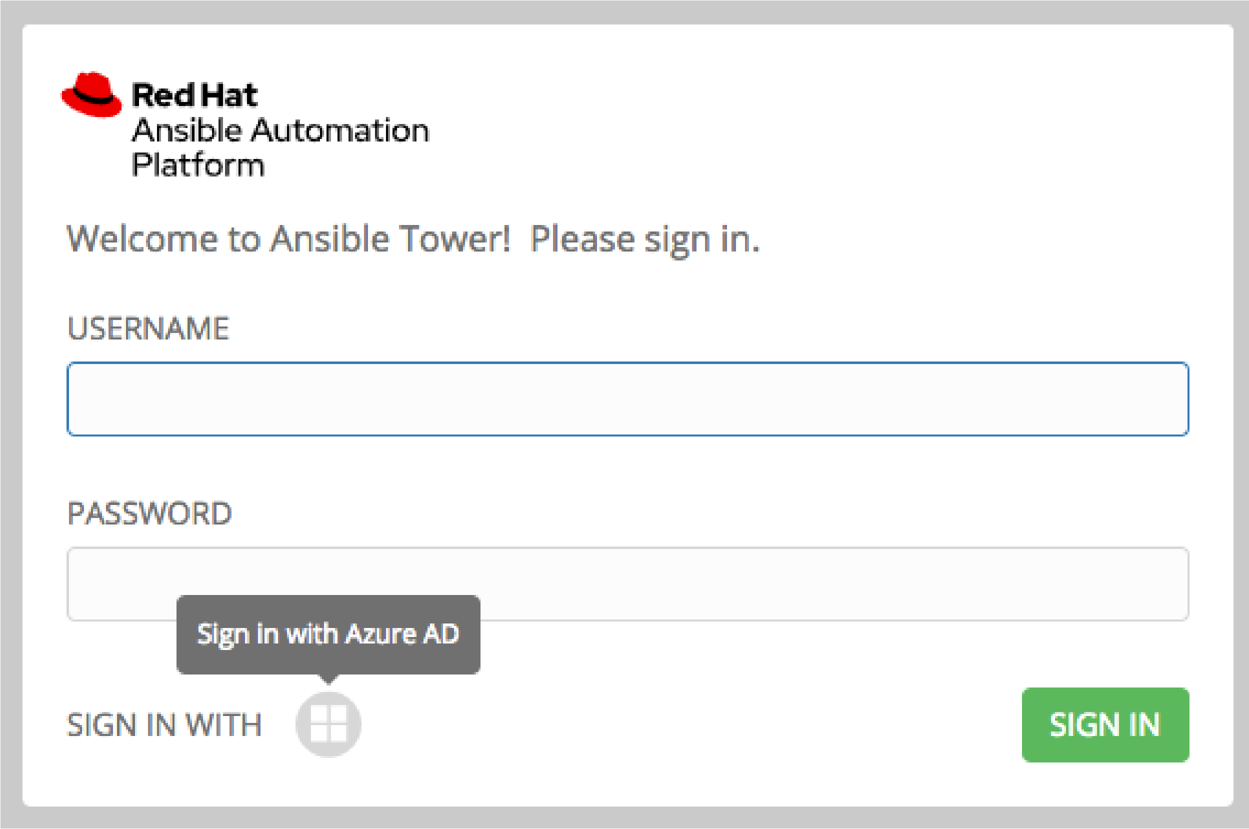
3. The **Azure AD OAuth2 Callback URL** field is already pre-populated and non-editable.

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

4. Copy and paste Azure's Application ID to the **Azure AD OAuth2 Key** field.

Following Azure AD's documentation for connecting your app to Microsoft Azure Active Directory, supply the key (shown at one time only) to the client for authentication.

5. Copy and paste the actual secret key created for your Azure AD application to the **Azure AD OAuth2 Secret** field of the Configure Tower - Authentication screen.
6. For details on completing the mapping fields, see *Organization and Team Mapping*.
7. Click **Save** when done.
8. To verify that the authentication was configured correctly, logout of Ansible Tower and the login screen will now display the Microsoft Azure logo to allow logging in with those credentials.



Red Hat Ansible Automation Platform

Welcome to Ansible Tower! Please sign in.

For application registering basics in Azure AD, refer to the Azure AD Identity Platform (v2) overview.

21.2 LDAP Authentication

Refer to the *Setting up LDAP Authentication* section.

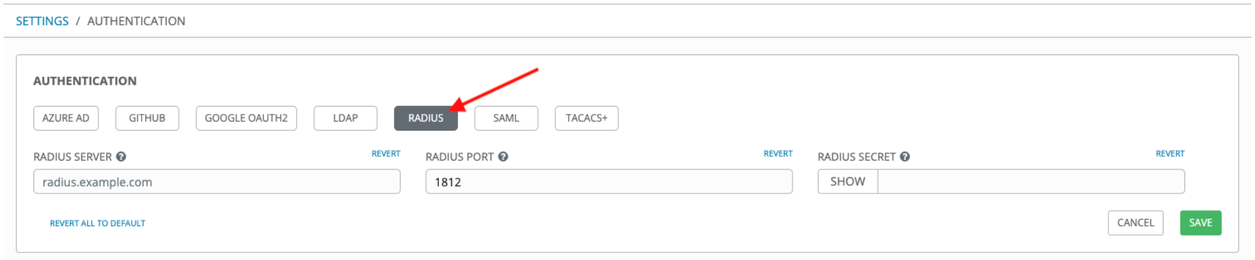
21.3 RADIUS Authentication Settings

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

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

The Azure AD tab displays initially by default.

2. Select the **Radius** tab.



RADIUS SERVER: radius.example.com

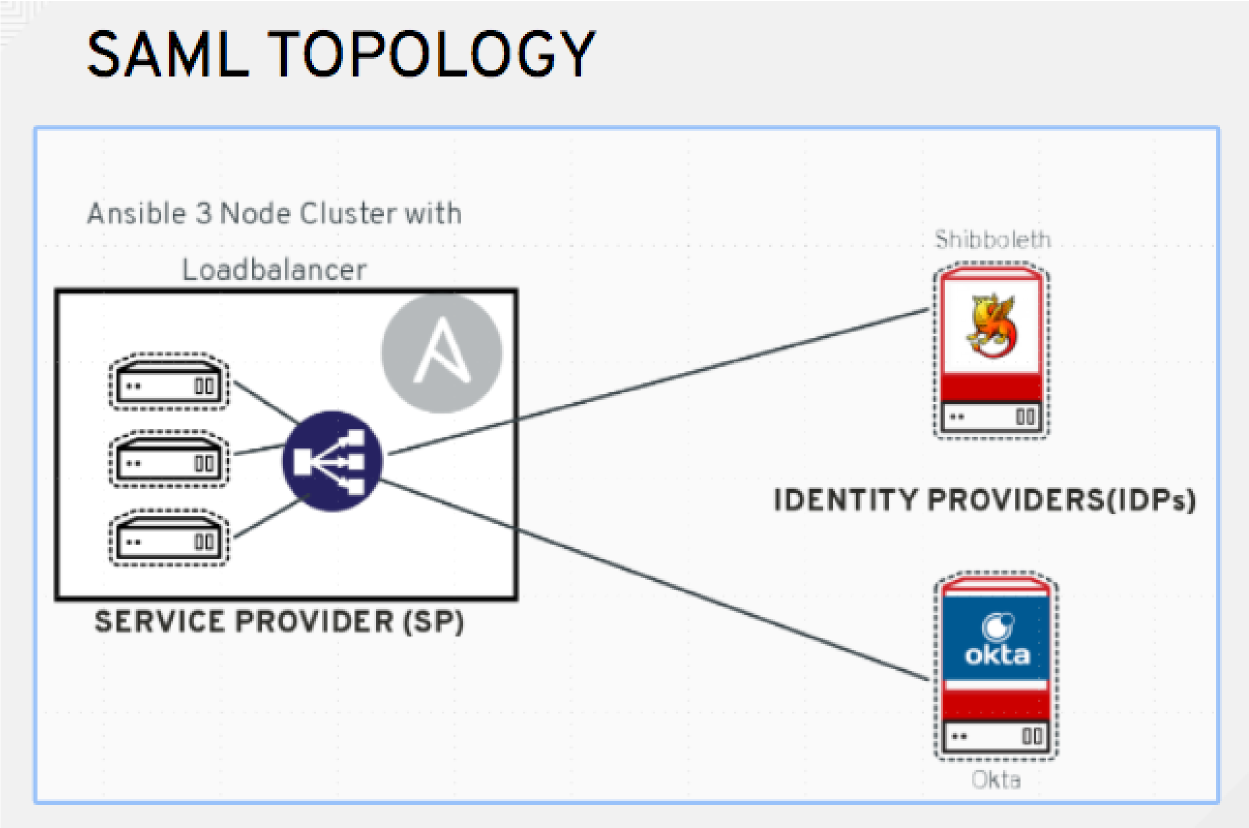
RADIUS PORT: 1812

RADIUS SECRET:

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

21.4 SAML Authentication Settings

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



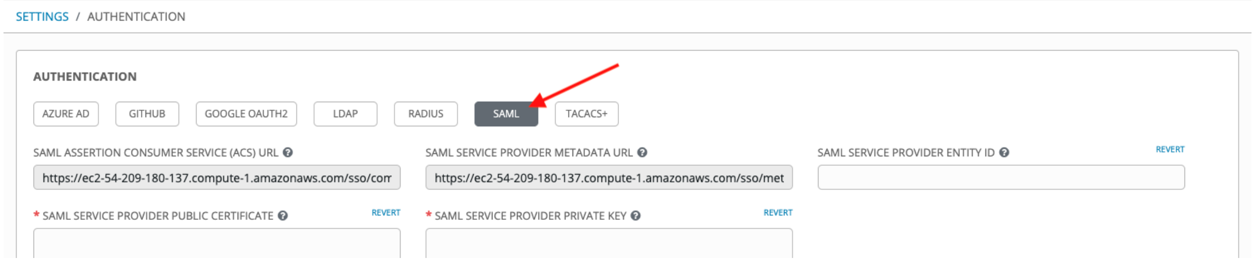
The following instructions describe Ansible Tower as the service provider. To authenticate users through RHSSO (keycloak), refer to the Red Hat Single Sign On Integration with Ansible Tower blog.

To setup SAML authentication:

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

The Azure AD tab displays initially by default.

2. Select the **SAML** tab.



SETTINGS / AUTHENTICATION

AUTHENTICATION

SAML ASSERTION CONSUMER SERVICE (ACS) URL: https://ec2-54-209-180-137.compute-1.amazonaws.com/sso/com

SAML SERVICE PROVIDER METADATA URL: https://ec2-54-209-180-137.compute-1.amazonaws.com/sso/met

SAML SERVICE PROVIDER ENTITY ID:

* SAML SERVICE PROVIDER PUBLIC CERTIFICATE:

* SAML SERVICE PROVIDER PRIVATE KEY:

The following steps describe **all** the fields and what they are used for. To get transparent SAML logins functional, the minimum required fields are marked with an asterisk (*) in the user interface.

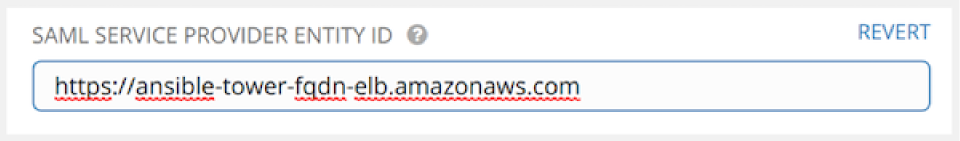
3. The **SAML Assertion Consume Service (ACS) URL** and **SAML Service Provider Metadata URL** fields are pre-populated and are non-editable. Contact the Identity Provider administrator and provide the information contained in these fields.

4. Set the **SAML Service Provider Entity ID** to be the same as the Tower Base URL. The Tower Base URL can be

found in the **System** tab of the Configure Tower screen, which you can access through the Settings icon. Through the API, it can be viewed in the `/api/v2/settings/system`, under the `TOWER_URL_BASE` variable. The Entity ID can be set to any one of the individual Tower Cluster Nodes, but it is good practice to set it to the URL of the Service Provider. Ensure that the Base URL matches the FQDN of the load balancer (if used).

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

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

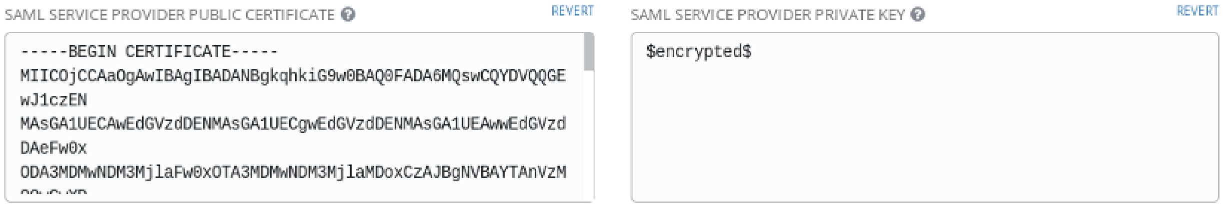


https://ansible-tower-fqdn-elb.amazonaws.com

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

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

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


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

```
-----BEGIN CERTIFICATE-----
MIICOjCCAaOgAwIBAgIBADANBgkqhkiG9w0BAQ0FADA6MQswCQYDVQQGE
wJ1czEN
MAsGA1UECAwEdGVzdDENMAsGA1UECgwEdGVzdDENMAsGA1UEAwwEdGVzd
DAeFw0x
ODA3MDMwNDM3MjlaFw0xOTA3MDMwNDM3MjlaMDoxCzAJBgNVBAYTAnVzM
```

```
$encrypted$
```

As an example for public certs:

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

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

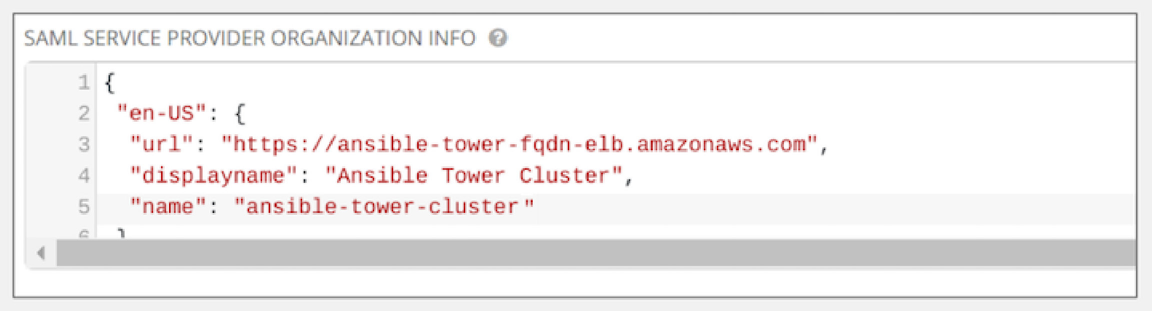
As an example for private keys:

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

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

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

For example:



SAML SERVICE PROVIDER ORGANIZATION INFO

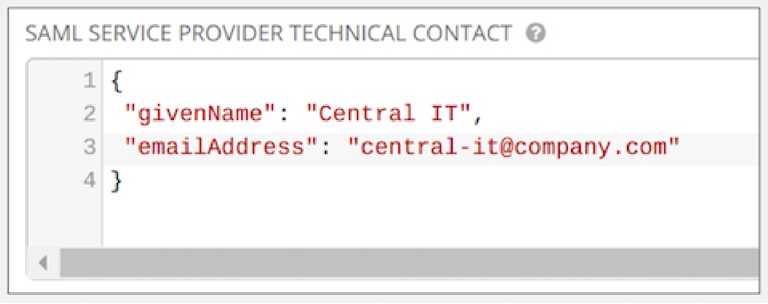
```
{
 "en-US": {
  "url": "https://ansible-tower-fqdn-elb.amazonaws.com",
  "displayname": "Ansible Tower Cluster",
  "name": "ansible-tower-cluster"
 }
```

Note: These fields are required in order to properly configure SAML within Tower.

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

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

For example:



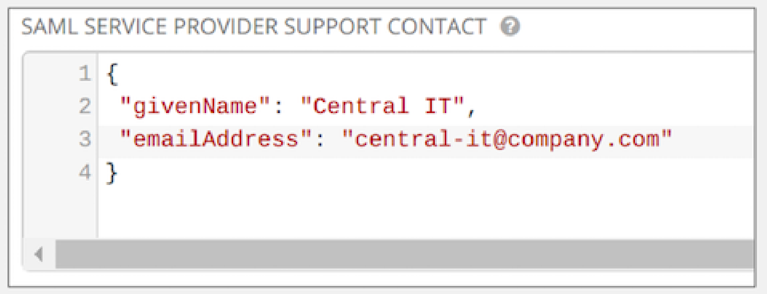
SAML SERVICE PROVIDER TECHNICAL CONTACT

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

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

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

For example:


SAML SERVICE PROVIDER SUPPORT CONTACT

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

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

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

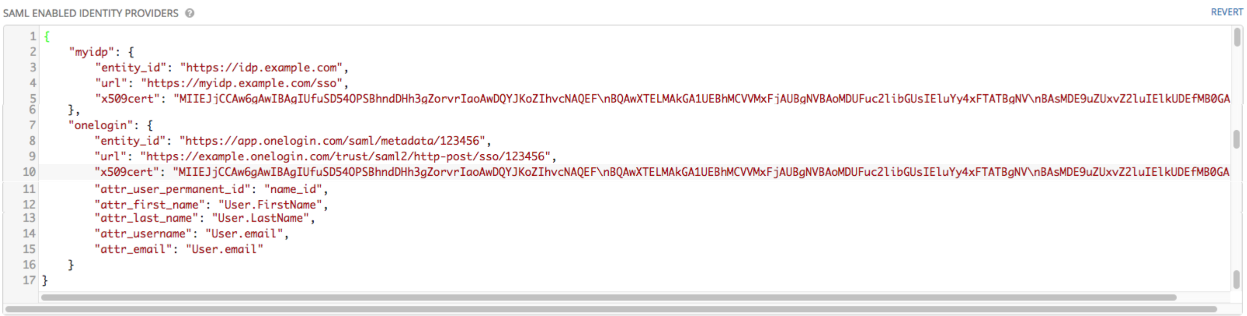
If these attributes are not known, map existing SAML attributes to lastname, firstname, email and username.

Configure the required keys for each IDp:

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

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

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



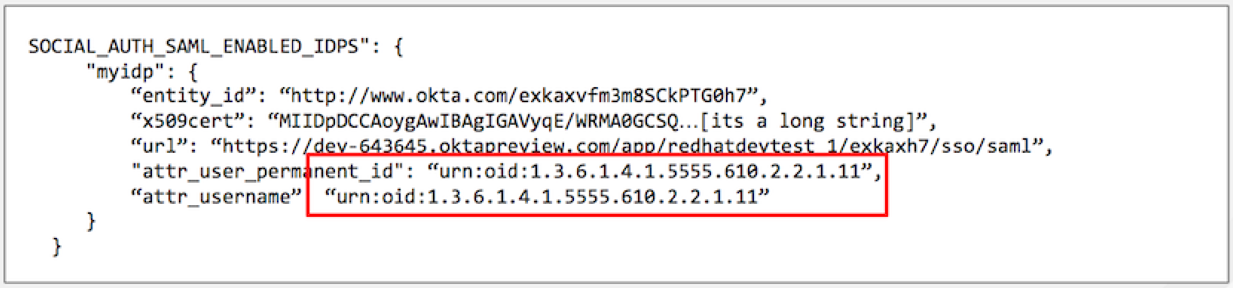
SAML ENABLED IDENTITY PROVIDERS

REVERT

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

Warning: `attr_username` must reference a unique per-account attribute. For example, if `attr_username` references `User.email` and a SAML user that shares the same email with another user (including a non-SAML user), the duplicated email accounts will be merged. Be aware that this same behavior exists for System Admin users, thus a SAML login with the same email address as the System Admin user will login with System Admin privileges. For future reference, you can remove (or add) Admin Privileges based on SAML mappings, as described in subsequent steps.

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



```
SOCIAL_AUTH_SAML_ENABLED_IDPS": {
    "myidp": {
        “entity_id”: “http://www.okta.com/exkaxvfm3m8SCkPTG0h7”,
        “x509cert”: “MIIDpDCCAoygAwIBAgIGAVyqE/WRMA0GCSQ...[its a long string]”,
        “url”: “https://dev-643645.oktapreview.com/app/redhatdevtest_1/exkaxh7/sso/saml”,
        "attr_user_permanent_id": “urn:oid:1.3.6.1.4.1.5555.610.2.2.1.11”,
        “attr_username”  “urn:oid:1.3.6.1.4.1.5555.610.2.2.1.11”
    }
}
```

11. Optionally provide the **SAML Organization Map**. For further detail, see *Organization and Team Mapping*.

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

Example SAML Organization Attribute Mapping

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

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

(continues on next page)

(continued from previous page)

```
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
        <saml2:AttributeValue>Engineering</saml2:AttributeValue>
    </saml2:Attribute>
</saml2:AttributeStatement>
```

Below is the corresponding Tower configuration.

```
{
  "saml_attr": "member-of",
  "saml_admin_attr": "admin-of",
  "remove": true,
  "remove_admins": false
}
```

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

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

Example SAML Team Attribute Mapping

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

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

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

- `saml_attr`: The SAML attribute name where the team array can be found.
- `remove`: Set `remove` to **True** to remove user from all Teams before adding the user to the list of Teams. To keep the user in whatever Team(s) they are in while adding the user to the Team(s) in the SAML attribute, set

remove to **False**.

- team_org_map: An array of dictionaries of the form { "team": "<AWX Team Name>", "organization": "<AWX Org Name>" } that defines mapping from Tower Team -> Tower Organization. This is needed because the same named Team can exist in multiple Organizations in Tower. The organization to which a team listed in a SAML attribute belongs to, would be ambiguous without this mapping.

You could create an alias to override both Teams and Orgs in the **SAML Team Attribute Mapping**. This option becomes very handy in cases when the SAML backend sends out complex group names, like in the example below:

```
{
 "remove": false,
 "team_org_map": [
  {
   "team": "internal:unix:domain:admins",
   "organization": "Default",
   "team_alias": "Administrators"
  },
  {
   "team": "Domain Users",
   "organization_alias": "OrgAlias",
   "organization": "Default"
  }
 ],
 "saml_attr": "member-of"
}
```

Once the user authenticates, Tower creates organization and team aliases, as expected.

13. Optionally provide team membership mapping in the **SAML Team Map** field. For further detail, see *Organization and Team Mapping*.

14. Optionally provide security settings in the **SAML Security Config** field. This field is the equivalent to the SOCIAL_AUTH_SAML_SECURITY_CONFIG field in the API. Refer to the OneLogin's SAML Python Toolkit for further detail.

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

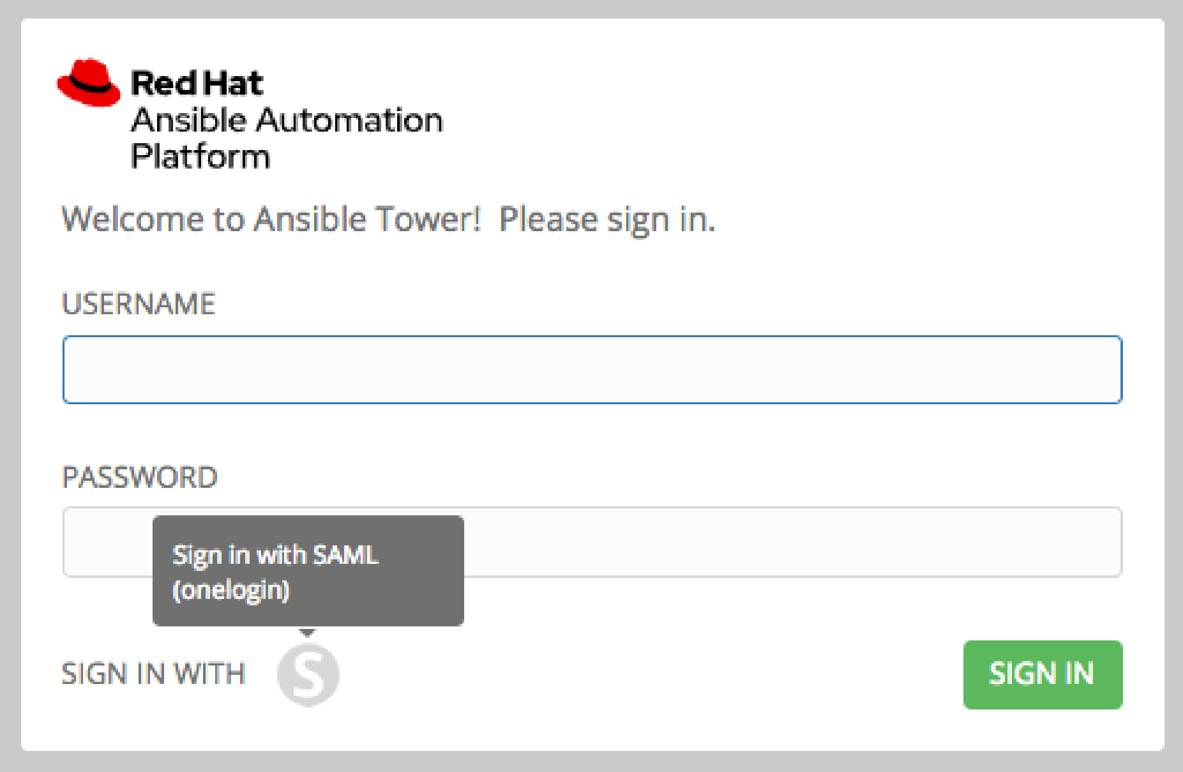
15. The **SAML Service Provider Extra Configuration Data** field is equivalent to the SOCIAL_AUTH_SAML_SP_EXTRA in the API. Refer to the python-saml library documentation to learn about the valid service provider extra (SP_EXTRA) parameters.

16. The **SAML IDP to EXTRA_DATA Attribute Mapping** field is equivalent to the SOCIAL_AUTH_SAML_EXTRA_DATA in the API. See Python's SAML Advanced Settings documentation for more information.

17. Click **Save** when done.

18. To verify that the authentication was configured correctly, load the auto-generated URL found in the **SAML Service Provider Metadata URL** into a browser. It should output XML output, otherwise, it is not configured correctly.

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



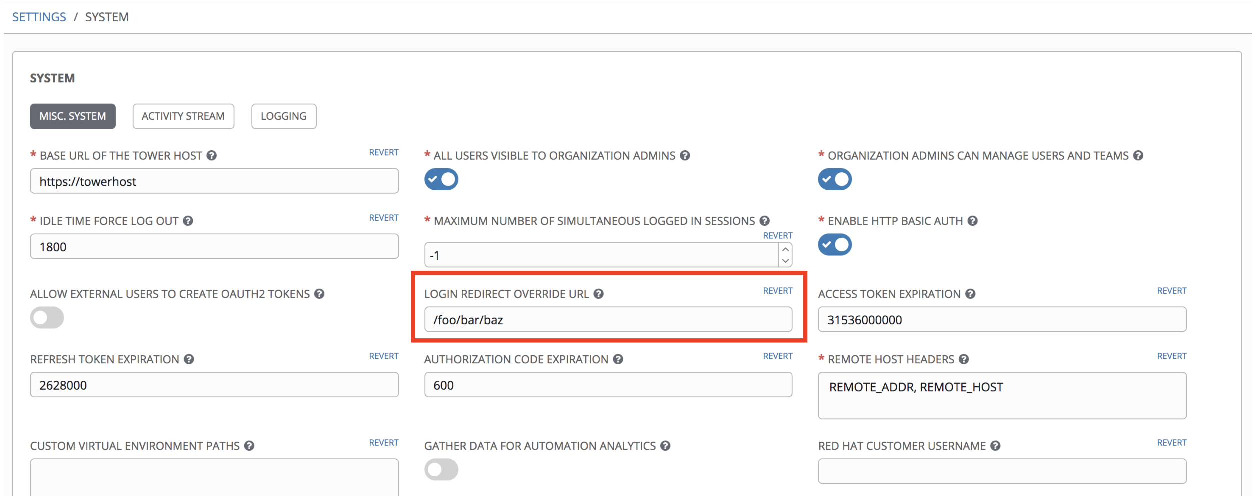
Welcome to Ansible Tower! Please sign in.

21.4.1 Transparent SAML Logins

For transparent logins to work, you must first get IdP-initiated logins to work. To achieve this:

1. Set the `RelayState` on the IdP to the key of the IdP definition in the `SAML Enabled Identity Providers` field as previously described. In the example given above, `RelayState` would need to be either `myidp` or `onelogin`.

2. Once this is working, in the Systems window of the Settings (⚙) menu of the Ansible Tower User Interface, use the **Login Redirect Override URL** field to specify the redirect URL for non-logged-in users to somewhere other than the default Tower login page. This should be set to `/sso/login/saml/?idp=<name-of-your-idp>` for transparent SAML login, as shown in the example.



SETTINGS / SYSTEM

SYSTEM

* BASE URL OF THE TOWER HOST: https://towerhost

* ALL USERS VISIBLE TO ORGANIZATION ADMINS: on

* ORGANIZATION ADMINS CAN MANAGE USERS AND TEAMS: on

* IDLE TIME FORCE LOG OUT: 1800

* MAXIMUM NUMBER OF SIMULTANEOUS LOGGED IN SESSIONS: -1

* ENABLE HTTP BASIC AUTH: on

ALLOW EXTERNAL USERS TO CREATE OAUTH2 TOKENS: off

LOGIN REDIRECT OVERRIDE URL: /foo/bar/baz

ACCESS TOKEN EXPIRATION: 31536000000

REFRESH TOKEN EXPIRATION: 2628000

AUTHORIZATION CODE EXPIRATION: 600

* REMOTE HOST HEADERS: REMOTE_ADDR, REMOTE_HOST

CUSTOM VIRTUAL ENVIRONMENT PATHS:

GATHER DATA FOR AUTOMATION ANALYTICS: off

RED HAT CUSTOMER USERNAME:

Note: The above is a sample of a typical IdP format, but may not be the correct format for your particular case. You may need to reach out to your IdP for the correct transparent redirect URL as that URL is not the same for all IdPs.

---

3. After transparent SAML login is configured, to log in using local credentials or a different SSO, go directly to `https://<your-tower-server>/login`. This provides the standard Tower login page, including SSO authentication buttons, and allows you to log in with any configured method.

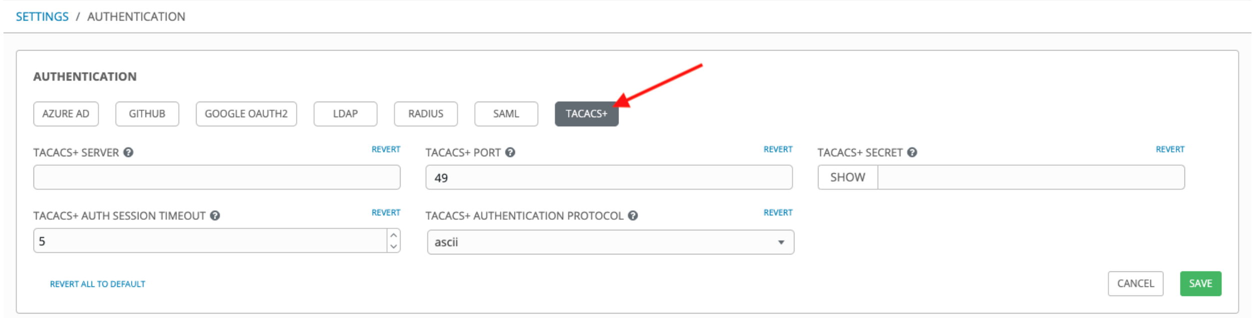
21.5 TACACS+ Authentication Settings

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

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

The Azure AD tab displays initially by default.

2. Select the **TACACs+** tab.



SETTINGS / AUTHENTICATION

AUTHENTICATION

TACACS+ SERVER:

TACACS+ PORT: 49

TACACS+ SECRET:

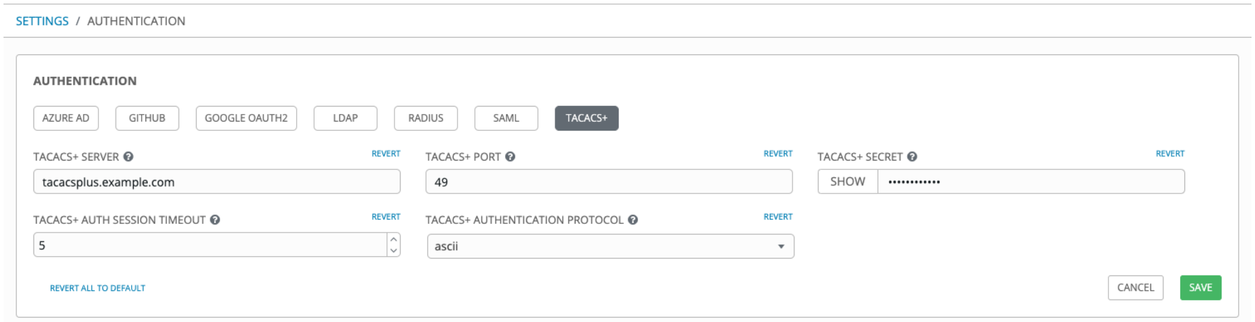
TACACS+ AUTH SESSION TIMEOUT: 5

TACACS+ AUTHENTICATION PROTOCOL: ascii

3. Enter information in the following fields:

- **TACACS+ Server**: Provide the hostname or IP address of the TACACS+ server with which to authenticate. If this field is left blank, TACACS+ authentication is disabled.

- **TACACS+ Port**: TACACS+ uses port 49 by default, which is already pre-populated.
- **TACACS+ Secret**: Secret key for TACACS+ authentication server.
- **TACACS+ Auth Session Timeout**: Session timeout value in seconds. The default is 5 seconds.
- **TACACS+ Authentication Protocol**: The protocol used by TACACS+ client. Options are **ascii** or **pap**.



TACACS+ SERVER: tacacsplus.example.com

TACACS+ PORT: 49

TACACS+ SECRET: ••••••••••••

TACACS+ AUTH SESSION TIMEOUT: 5

TACACS+ AUTHENTICATION PROTOCOL: ascii

4. Click **Save** when done.

CHAPTER

TWENTYTWO

SETTING UP LDAP AUTHENTICATION

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

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

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

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

To configure LDAP integration for Tower:

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

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

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

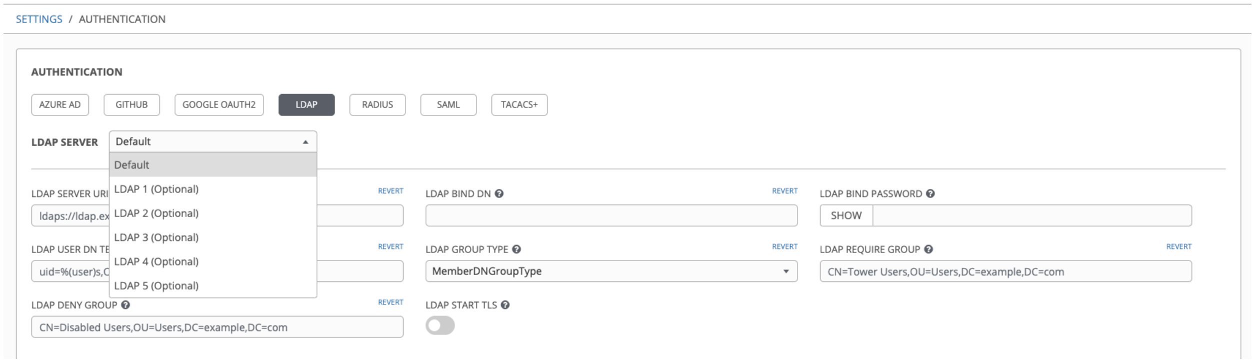
Note: The `ldapsearch` utility is not automatically pre-installed with Ansible Tower, however, you can install it from the `openldap-clients` package.

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

The Azure AD tab displays initially by default.

4. Select the **LDAP** tab.

Note: You can configure multiple LDAP servers by specifying the server to configure (otherwise, leave the server at **Default**):



LDAP SERVER: Default

LDAP SERVER URI: ldaps://ldap.ex

LDAP BIND DN:

LDAP BIND PASSWORD:

LDAP USER DN TE: uid=%(user)s,C

LDAP GROUP TYPE: MemberDNGroupType

LDAP REQUIRE GROUP: CN=Tower Users,OU=Users,DC=example,DC=com

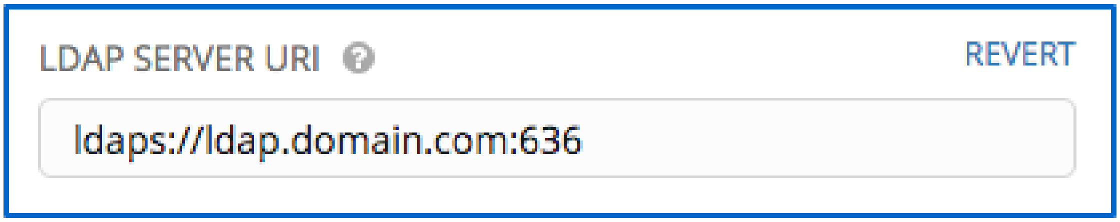
LDAP DENY GROUP: CN=Disabled Users,OU=Users,DC=example,DC=com

LDAP START TLS: ○

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

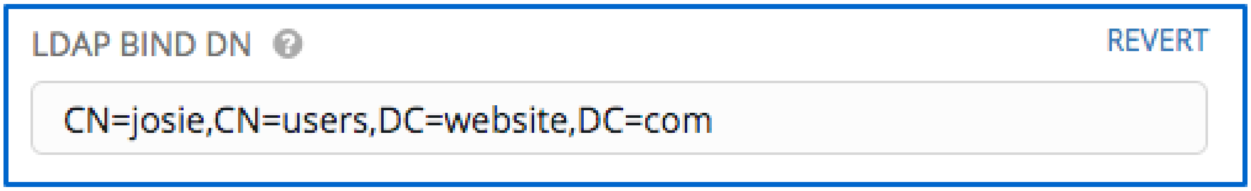
---

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



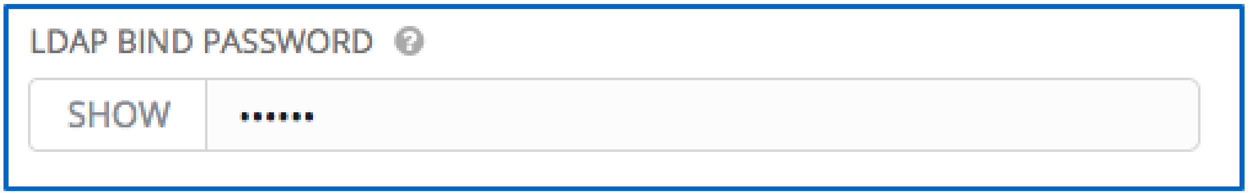
LDAP SERVER URI: ldaps://ldap.domain.com:636

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



LDAP BIND DN: CN=josie,CN=users,DC=website,DC=com

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



LDAP BIND PASSWORD: ••••••

8. If that name is stored in key `sAMAccountName`, the **LDAP User DN Template** populates with `(sAMAccountName=%(user)s)`. Active Directory stores the username to `sAMAccountName`. Similarly, for OpenLDAP, the key is `uid`–hence the line becomes `(uid=%(user)s)`.

9. Click to select a group type from the **LDAP Group Type** drop-down menu list.

LDAP Group Types include:

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

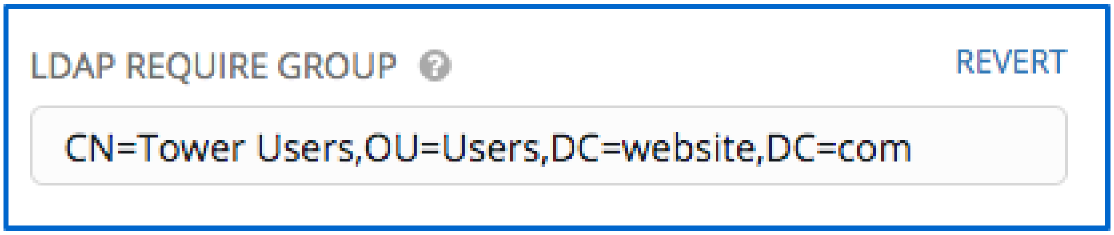
The LDAP Group Types that are supported by Tower leverage the underlying django-auth-ldap library.

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

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

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

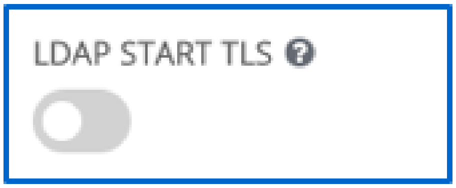
10. Enter the group distinguish name to allow users within that group to access Tower in the **LDAP Require Group** field, using the same format as the one shown in the text field. In this example, use: CN=Tower Users, OU=Users,DC=website,DC=com



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

11. Enter the group distinguish name to prevent users within that group to access Tower in the **LDAP Deny Group** field, using the same format as the one shown in the text field. In this example, leave the field blank.

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



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

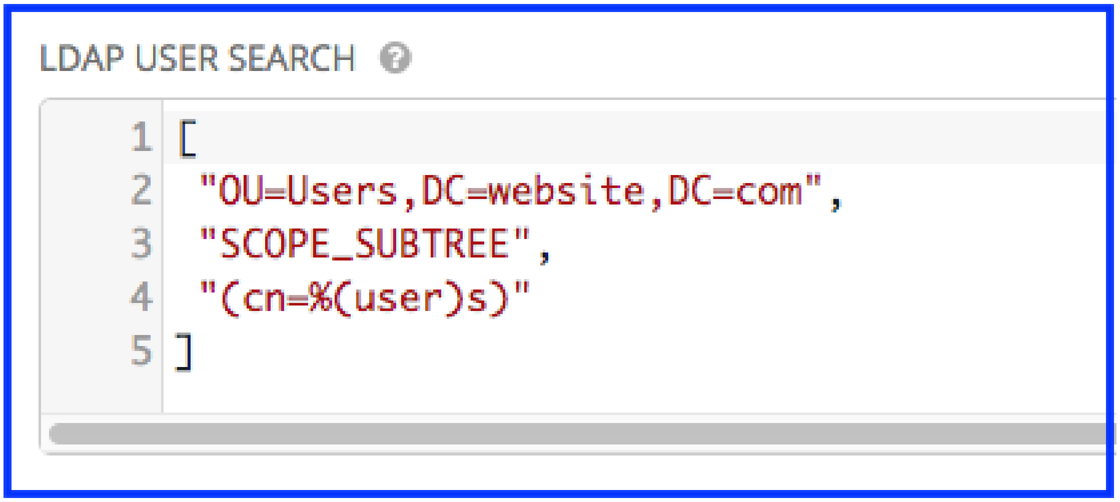
```
[
"OU=Users,DC=website,DC=com",
"SCOPE_SUBTREE",
"(cn=%(user)s)"
]
```

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

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

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

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



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

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

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

(continues on next page)

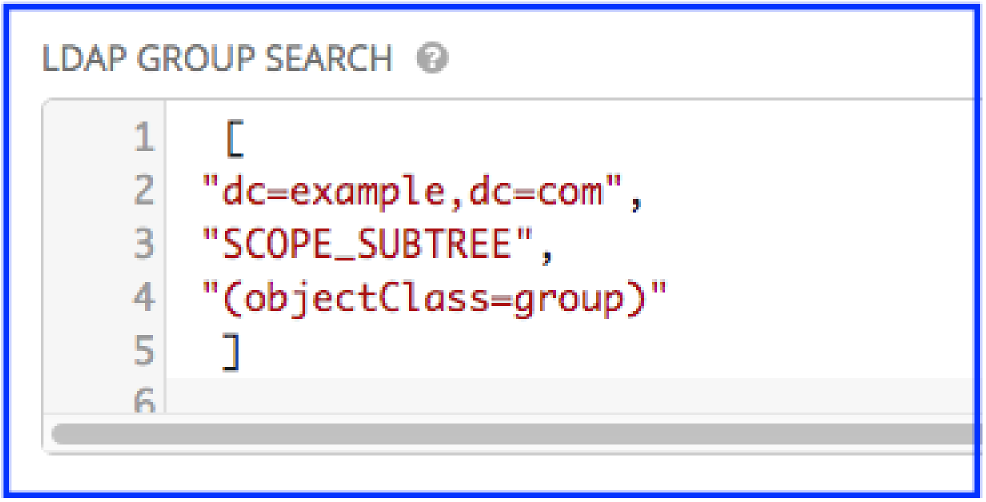
(continued from previous page)

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

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

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

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

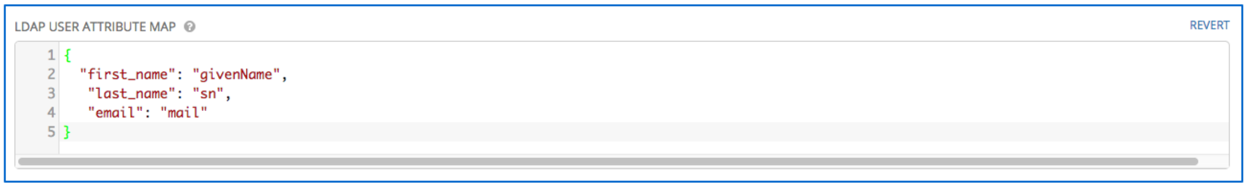


```
 [
"dc=example,dc=com",
"SCOPE_SUBTREE",
"(objectClass=group)"
 ]
```

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

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

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

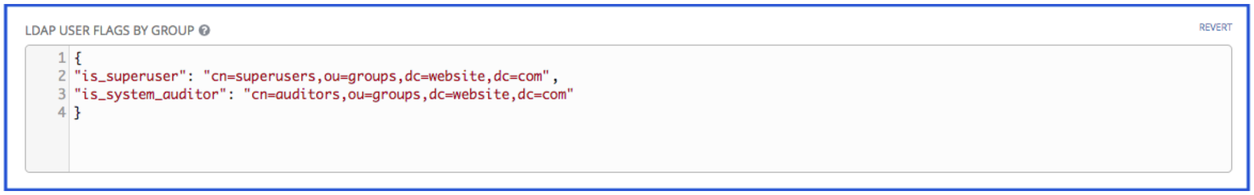


```
{
  "first_name": "givenName",
   "last_name": "sn",
   "email": "mail"
}
```

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

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

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



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

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

18. Click **Save** when done.

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

Note: Tower does not actively sync users, but they are created during their initial login. To improve performance associated with LDAP authentication, see ug_ldap_auth_perf_tips in the *Ansible Tower User Guide*.

22.1 Referrals

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

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

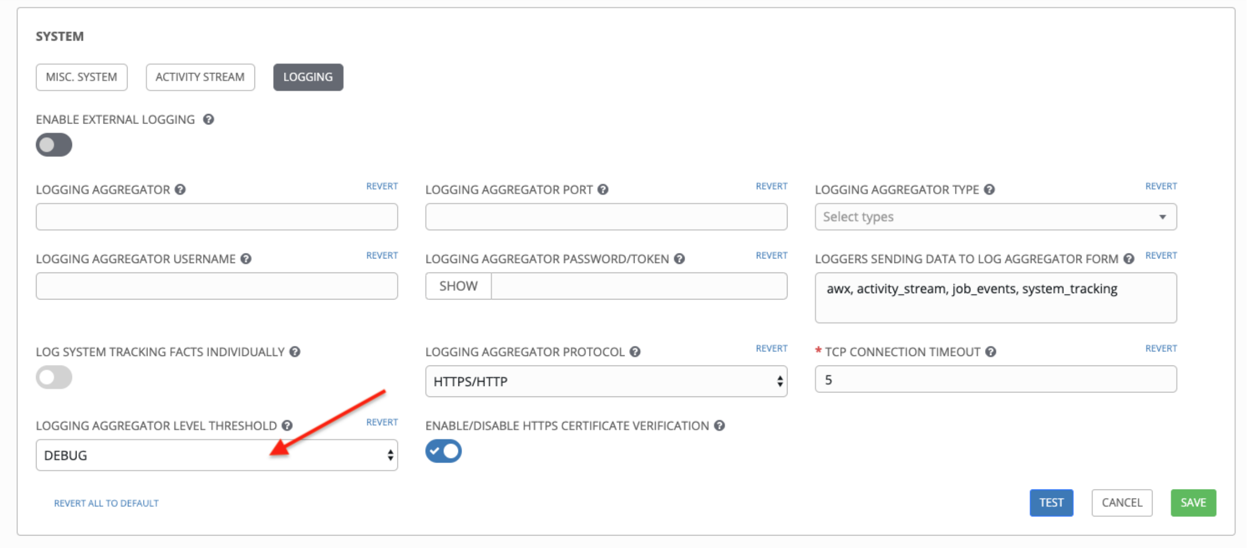
Note: "Referrals" are disabled by default in Ansible Tower version 2.4.3 and above. If you are running an earlier version of Tower, you should consider adding this parameter to your configuration file.

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

22.2 Enabling Logging for LDAP

To enable logging for LDAP, you must set the level to `DEBUG` in the Tower Settings configuration window:

1. Click the **Settings** (⚙) icon from the left navigation pane and select **System**.
2. From the System configuration page, click the **Logging** tab.
3. Scroll down to the bottom and set the **Logging Aggregator Level Threshold** field to **Debug**.



SYSTEM

MISC. SYSTEM | ACTIVITY STREAM | LOGGING

ENABLE EXTERNAL LOGGING: off

LOGGING AGGREGATOR: (empty)

LOGGING AGGREGATOR PORT: (empty)

LOGGING AGGREGATOR TYPE: Select types

LOGGING AGGREGATOR USERNAME: (empty)

LOGGING AGGREGATOR PASSWORD/TOKEN: SHOW

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

LOG SYSTEM TRACKING FACTS INDIVIDUALLY: off

LOGGING AGGREGATOR PROTOCOL: HTTPS/HTTP

* TCP CONNECTION TIMEOUT: 5

LOGGING AGGREGATOR LEVEL THRESHOLD: DEBUG

ENABLE/DISABLE HTTPS CERTIFICATE VERIFICATION: on

4. Click **Save** to save your changes.

22.3 LDAP Organization and Team Mapping

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

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

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

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

remove_admins: True/False. Defaults to False.

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

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

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

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

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

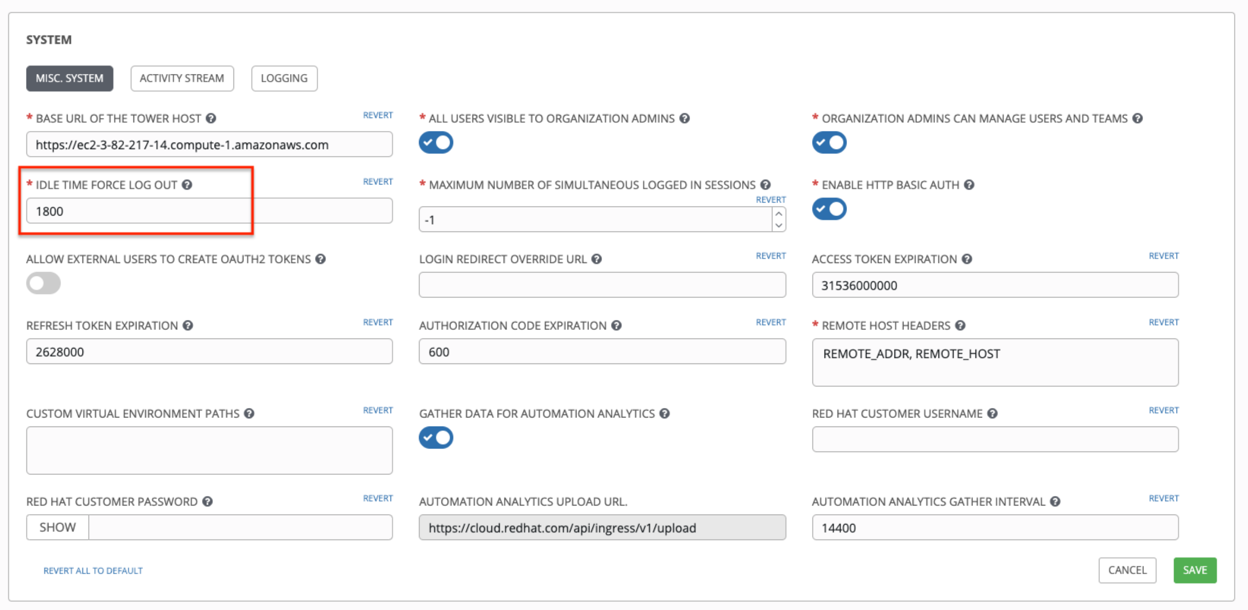
```
}
}
```

CHAPTER TWENTYTHREE

CHANGING THE DEFAULT TIMEOUT FOR AUTHENTICATION

The default length of time, in seconds, that your supplied token is valid can be changed in the System Settings screen of the Tower User Interface:

1. From the Settings (⚙) Menu screen, click **System**.
2. Select the **Misc. System** tab, if not already the default view.
3. Enter the timeout period in seconds in the **Idle Time Force Log Out** text field.



* BASE URL OF THE TOWER HOST: https://ec2-3-82-217-14.compute-1.amazonaws.com

* ALL USERS VISIBLE TO ORGANIZATION ADMINS: ● on

* ORGANIZATION ADMINS CAN MANAGE USERS AND TEAMS: ● on

* IDLE TIME FORCE LOG OUT: 1800

* MAXIMUM NUMBER OF SIMULTANEOUS LOGGED IN SESSIONS: -1

* ENABLE HTTP BASIC AUTH: ● on

ALLOW EXTERNAL USERS TO CREATE OAUTH2 TOKENS: ○ off

LOGIN REDIRECT OVERRIDE URL:

ACCESS TOKEN EXPIRATION: 31536000000

REFRESH TOKEN EXPIRATION: 2628000

AUTHORIZATION CODE EXPIRATION: 600

* REMOTE HOST HEADERS: REMOTE_ADDR, REMOTE_HOST

CUSTOM VIRTUAL ENVIRONMENT PATHS:

GATHER DATA FOR AUTOMATION ANALYTICS: ● on

RED HAT CUSTOMER USERNAME:

RED HAT CUSTOMER PASSWORD: SHOW

AUTOMATION ANALYTICS UPLOAD URL.: https://cloud.redhat.com/api/ingress/v1/upload

AUTOMATION ANALYTICS GATHER INTERVAL: 14400

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

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 TWENTYFOUR

USER AUTHENTICATION WITH KERBEROS

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

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

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

Once installed, edit the `/etc/krb5.conf` file, as follows, to provide the address of the AD, the domain, etc.:

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

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

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

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

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

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

Check if we got a valid ticket.

```

(continues on next page)

(continued from previous page)

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

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

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

```
[windows]
win01.WEBSITE.COM

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

You should also:

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

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

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

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

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

24.1 AD and Kerberos Credentials

Active Directory only:

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

With Kerberos:

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

24.2 Working with Kerberos Tickets

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

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

Automatic ticket management requires a standard kinit binary on the control host system path. To specify a different location or binary name, set the `ansible_winrm_kinit_cmd` inventory variable to the fully-qualified path to an MIT krbv5 kinit-compatible binary.

CHAPTER

TWENTYFIVE

WORKING WITH SESSION LIMITS

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

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

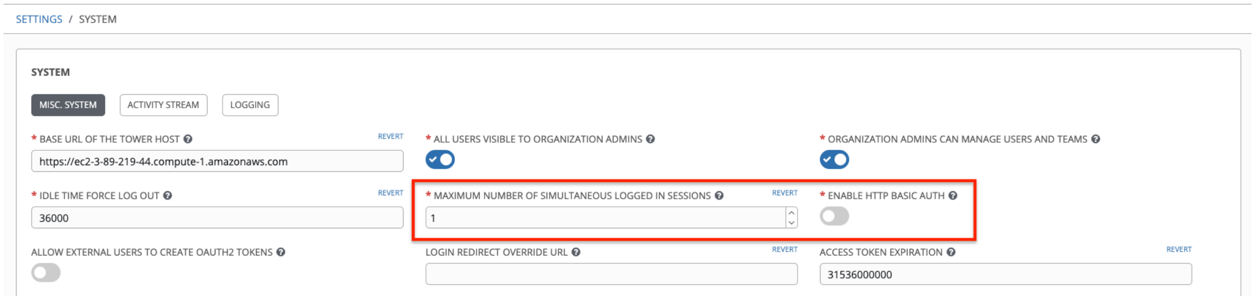
Session limits may be important, depending on your particular setup. For example, perhaps you only want a single user on your system with a single login per device (where the user could log in on his work laptop, phone, or home computer). In such a case, you would want to create a session limit equal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

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

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

To make changes to your session limits, navigate to *Configure Tower* and edit the **Maximum Number Of Simultaneous Logged In Sessions** setting or use the Browsable API if you are comfortable with making REST requests.

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. Alternatively, in the System Settings of the Tower UI, toggle the **Enable HHTP Basic Auth** to off.



* BASE URL OF THE TOWER HOST: https://ec2-3-89-219-44.compute-1.amazonaws.com

* ALL USERS VISIBLE TO ORGANIZATION ADMINS: ✓

* ORGANIZATION ADMINS CAN MANAGE USERS AND TEAMS: ✓

* IDLE TIME FORCE LOG OUT: 36000

* MAXIMUM NUMBER OF SIMULTANEOUS LOGGED IN SESSIONS: 1

* ENABLE HTTP BASIC AUTH: ○

ALLOW EXTERNAL USERS TO CREATE OAUTH2 TOKENS: ○

LOGIN REDIRECT OVERRIDE URL:

ACCESS TOKEN EXPIRATION: 31536000000

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 TWENTYSIX

BACKING UP AND RESTORING TOWER

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

Note: When restoring, be sure to restore to the same version from which it was backed up. However, you should always use the most recent minor version of a release to backup and/or restore your Tower installation version. For example, if the current version of Tower that you are on is 3.7.x, use only the 3.7.3 installer.

Also, backup and restore will *only* work on PostgreSQL versions supported by your current Ansible Tower version. For more information, see Requirements in the *Ansible Automation Platform Installation and Reference Guide*.

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

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

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

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

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

Backup files will be created on the same path that setup.sh script exists. It can be changed by specifying the following `EXTRA_VARS` :

```
root@localhost:~# ./setup.sh -e 'backup_dest=/path/to/backup_dir/' -b
```

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

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

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

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

26.1 Backup/Restore Playbooks

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

These playbooks serve two functions–backup and restore.

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

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

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

26.2 Backup and Restoration Considerations

- Disk Space: Review your disk space requirements to ensure you have enough room to backup configuration files, keys, and other relevant files, plus the database of the Tower installation.
- System Credentials: Confirm you have the system credentials you need when working with a local database or a remote database. On local systems, you may need root or `sudo` access, depending on how credentials were setup. On remote systems, you may need different credentials to grant you access to the remote system you are trying to backup or restore.
- You should always use the most recent minor version of a release to backup and/or restore your Tower installation version. For example, if the current version of Tower that you are on is 3.7.x, use only the 3.7.3 installer.
- When using `setup.sh` to do a restore from the default restore file path, `/var/lib/awx`, `-r` is still required in order to do the restore, but it no longer accepts an argument. If a non-default restore file path is needed, the user must provide this as an extra var (`root@localhost:~# ./setup.sh -e 'restore_backup_file=/path/to/nondefault/backup.tar.gz' -r`).
- If the backup file is placed in the same directory as the `setup.sh` installer, the restore playbook will automatically locate the restore files. In this case, you do not need to use the `restore_backup_file` extra var to specify the location of the backup file.

26.3 Backup and Restore for Clustered Environments

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

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

When restoring to an existing cluster, the restore contains:

- Dump of the PostgreSQL database
- UI artifacts (included in database dump)
- Tower configuration (retrieved from `/etc/tower`)
- Tower secret key
- Manual projects

26.3.1 Restoring to a different cluster

When restoring a backup to a separate instance or cluster, manual projects and custom settings under /etc/tower are retained. Job output and job events are stored in the database, and therefore, not affected.

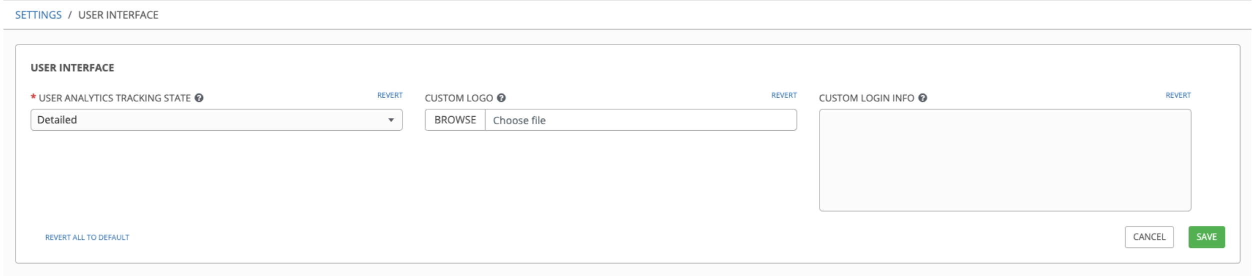
The restore process will not alter instance groups present before the restore (neither will it introduce any new instance groups). Restored Tower resources that were associated to instance groups will likely need to be reassigned to instance groups present on the new Tower cluster.

CHAPTER

TWENTYSEVEN

USING CUSTOM LOGOS IN ANSIBLE TOWER

Ansible Tower supports the use of a custom logo. You can add a custom logo by uploading an image; and supply a custom login message from the User Interface settings of the Settings (⚙) menu.


SETTINGS / USER INTERFACE

USER INTERFACE

* USER ANALYTICS TRACKING STATE: Detailed

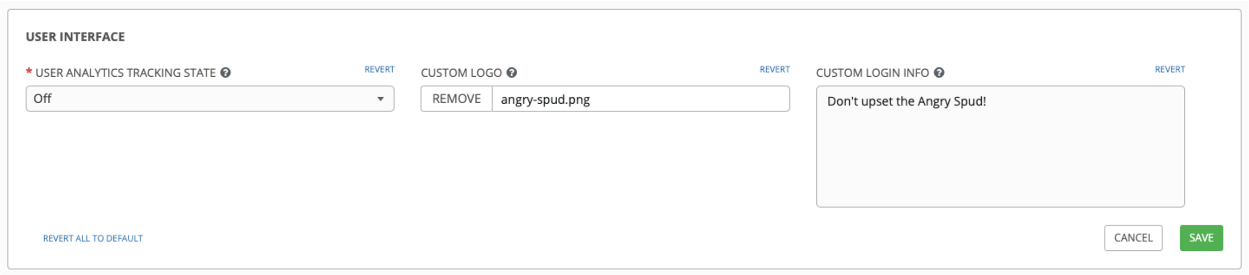
CUSTOM LOGO: Choose file

CUSTOM LOGIN INFO:

For the custom logo to look its best, use a `.png` file with a transparent background. GIF, PNG, and JPEG formats are supported.

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

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

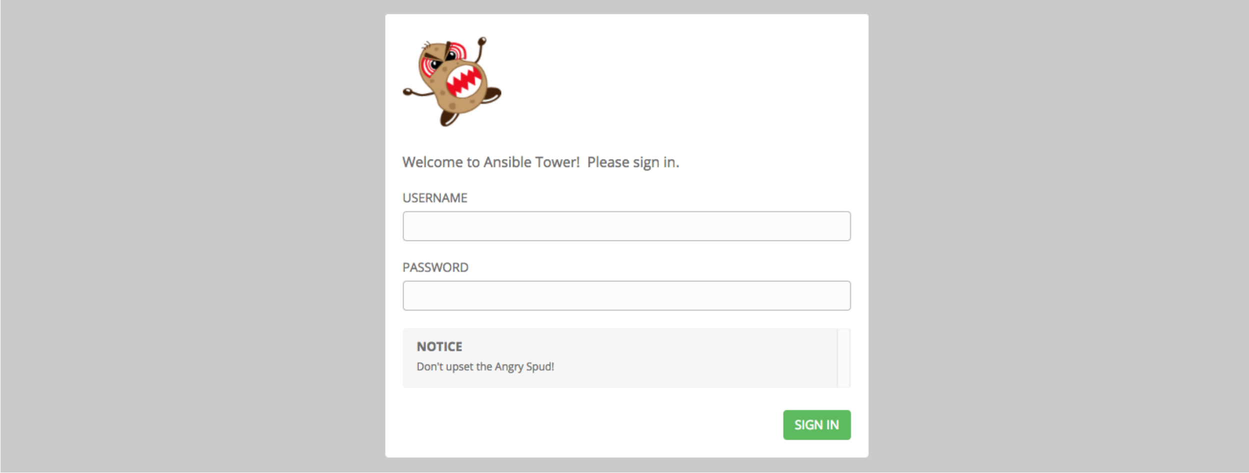

USER INTERFACE

* USER ANALYTICS TRACKING STATE: Off

CUSTOM LOGO: angry-spud.png

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

The Tower login dialog would look like this:



Welcome to Ansible Tower! Please sign in.

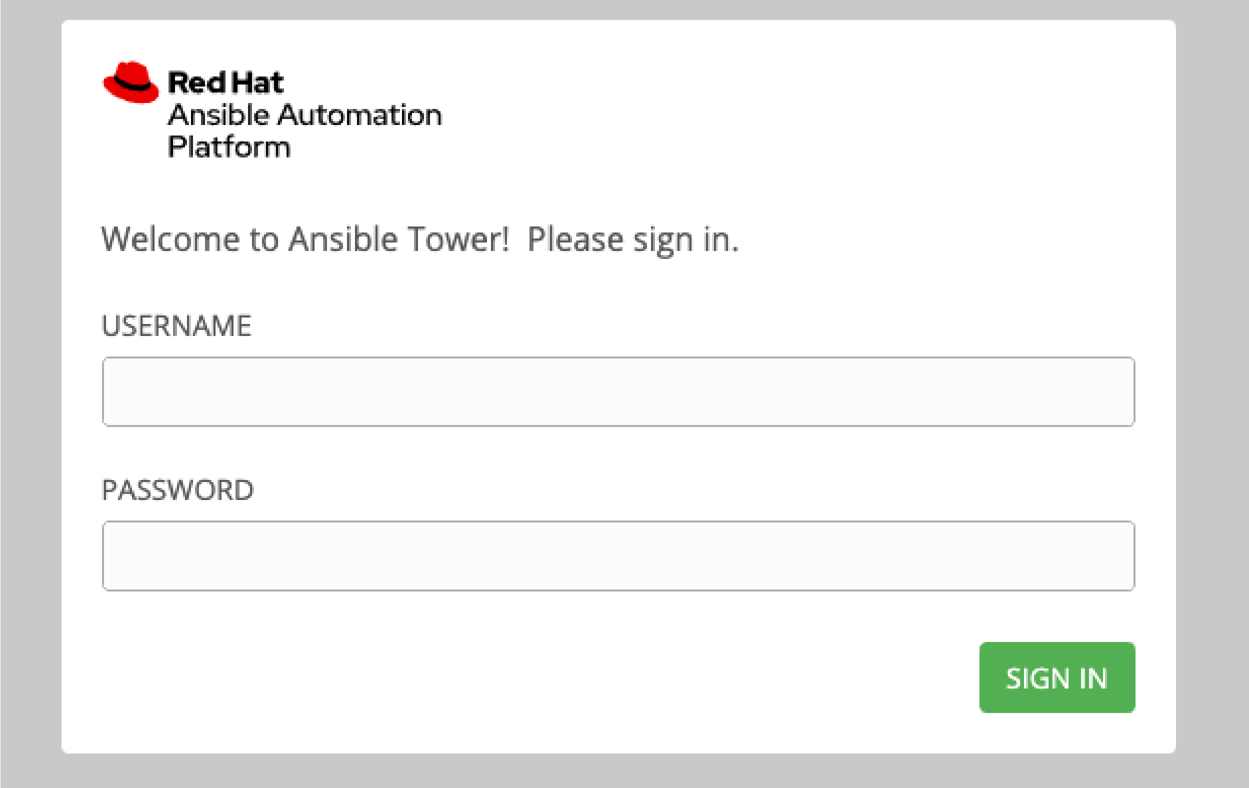
USERNAME

PASSWORD

NOTICE

Don't upset the Angry Spud!

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



Welcome to Ansible Tower! Please sign in.

USERNAME

PASSWORD

CHAPTER

TWENTYEIGHT

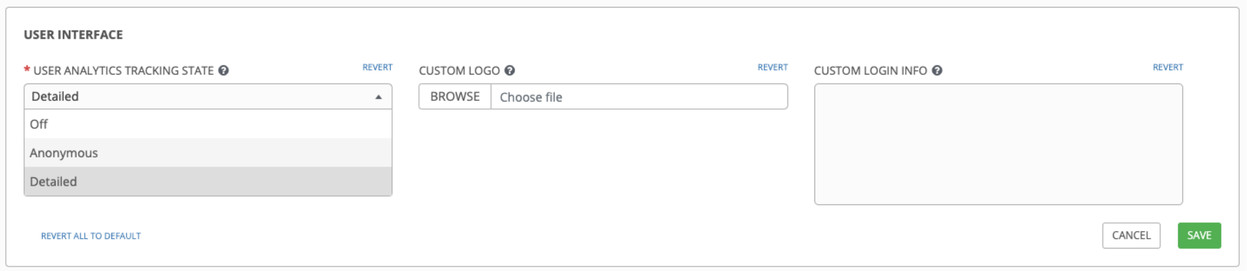
USABILITY ANALYTICS AND DATA COLLECTION

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

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

If you want to change how you participate in this analytics collection, you can opt out or change your settings using the Configure Tower user interface, accessible from the Settings (⚙) icon from the left navigation bar.

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



USER INTERFACE

* USER ANALYTICS TRACKING STATE: Detailed

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

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

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

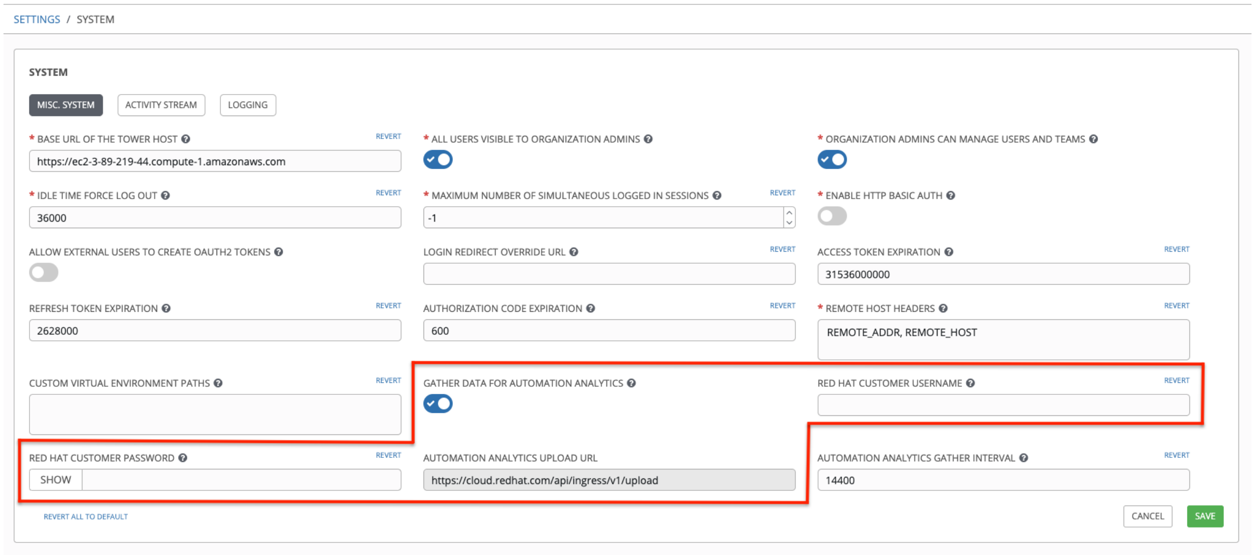
For more information, see the Red Hat privacy policy at https://www.redhat.com/en/about/privacy-policy.

28.1 Automation Analytics

When you imported your license for the first time, you were given options related to the collection of data that powers Automation Analytics, a cloud service that is part of the Ansible Automation Platform subscription. For opt-in of Automation Analytics to have any effect, your instance of Ansible Tower **must** be running on Red Hat Enterprise Linux.

Much like Red Hat Insights, Automation Analytics is built to only collect the minimum amount of data needed. No credential secrets, personal data, automation variables, or task output is gathered. For more information, see *Details of data collection* below.

In order to enable this feature, turn on data collection for Automation Analytics and enter your Red Hat customer credentials in the **Misc. System** tab of the **System** configuration window located in the Settings menu.



SETTINGS / SYSTEM

SYSTEM

* BASE URL OF THE TOWER HOST: https://ec2-3-89-219-44.compute-1.amazonaws.com

* ALL USERS VISIBLE TO ORGANIZATION ADMINS: ☑

* ORGANIZATION ADMINS CAN MANAGE USERS AND TEAMS: ☑

* IDLE TIME FORCE LOG OUT: 36000

* MAXIMUM NUMBER OF SIMULTANEOUS LOGGED IN SESSIONS: -1

* ENABLE HTTP BASIC AUTH: □

ALLOW EXTERNAL USERS TO CREATE OAUTH2 TOKENS: □

LOGIN REDIRECT OVERRIDE URL:

ACCESS TOKEN EXPIRATION: 31536000000

REFRESH TOKEN EXPIRATION: 2628000

AUTHORIZATION CODE EXPIRATION: 600

* REMOTE HOST HEADERS: REMOTE_ADDR, REMOTE_HOST

CUSTOM VIRTUAL ENVIRONMENT PATHS:

GATHER DATA FOR AUTOMATION ANALYTICS: ☑

RED HAT CUSTOMER USERNAME:

RED HAT CUSTOMER PASSWORD: SHOW

AUTOMATION ANALYTICS UPLOAD URL: https://cloud.redhat.com/api/ingress/v1/upload

AUTOMATION ANALYTICS GATHER INTERVAL: 14400

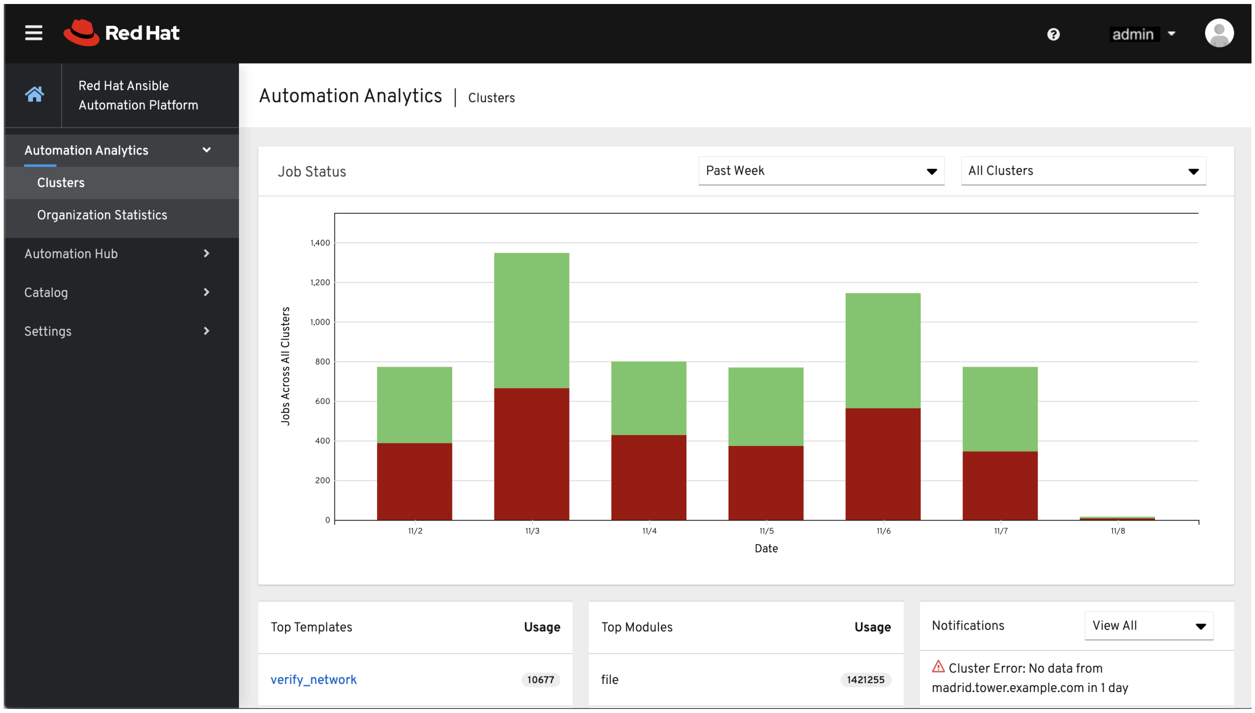
Note that the **Automation Analytics Upload URL** field is pre-populated with the location to which the collection of insights data will be uploaded.

By default, the Automation Analytics data is collected every 4 hours and upon enabling the feature, data will be collected up to a month back (or until the previous collection). You may turn off this data collection at any time in the **Misc. System** tab of the **System** configuration window.

This setting can also be enabled via the API by specifying `INSIGHTS_TRACKING_STATE = True` in either of these endpoints:

- `api/v2/settings/all`
- `api/v2/settings/system`

The Automation Analytics generated from this data collection will be found on the Red Hat Cloud Services portal.



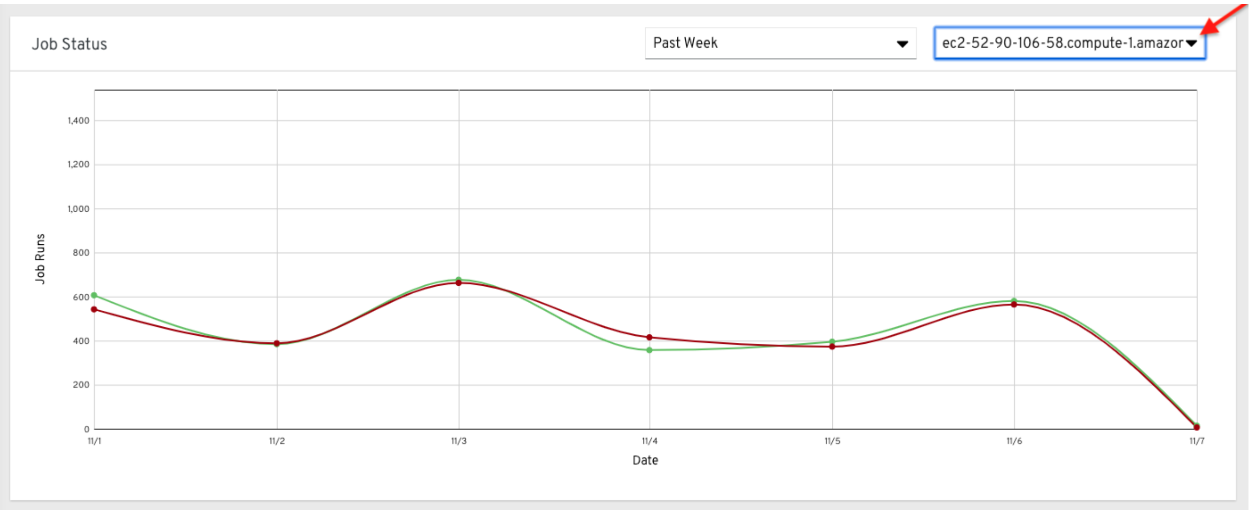
Top Templates	Usage
verify_network	10677

Top Modules	Usage
file	1421255

⚠ Cluster Error: No data from madrid.tower.example.com in 1 day

The **Clusters** data is the default view. This graph represents the number of job runs across all Tower clusters over a period of time. The example above shows a span of a week in a stacked bar-style chart that is organized by the number of jobs that ran successfully (in green) and jobs that failed (in red).

Alternatively, you can select a single cluster to view its job status information.

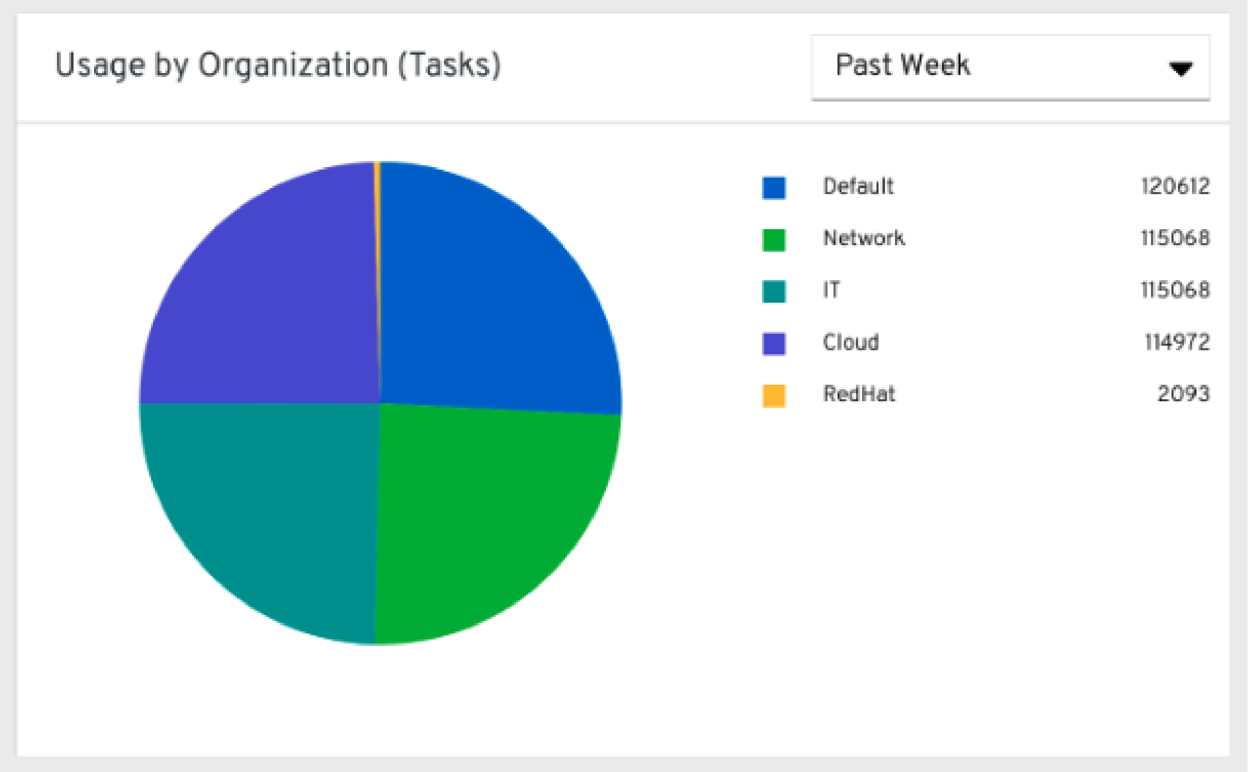


This multi-line chart represents the number of job runs for a single Tower cluster for a specified period of time. The example here shows a span of a week, organized by the number of successfully running jobs (in green) and jobs that failed (in red). You can specify the number of successful and failed job runs for a selected cluster over a span of one week, two weeks, and monthly increments.

Click **Organization Statistics** from the left navigation pane to view information for the following:

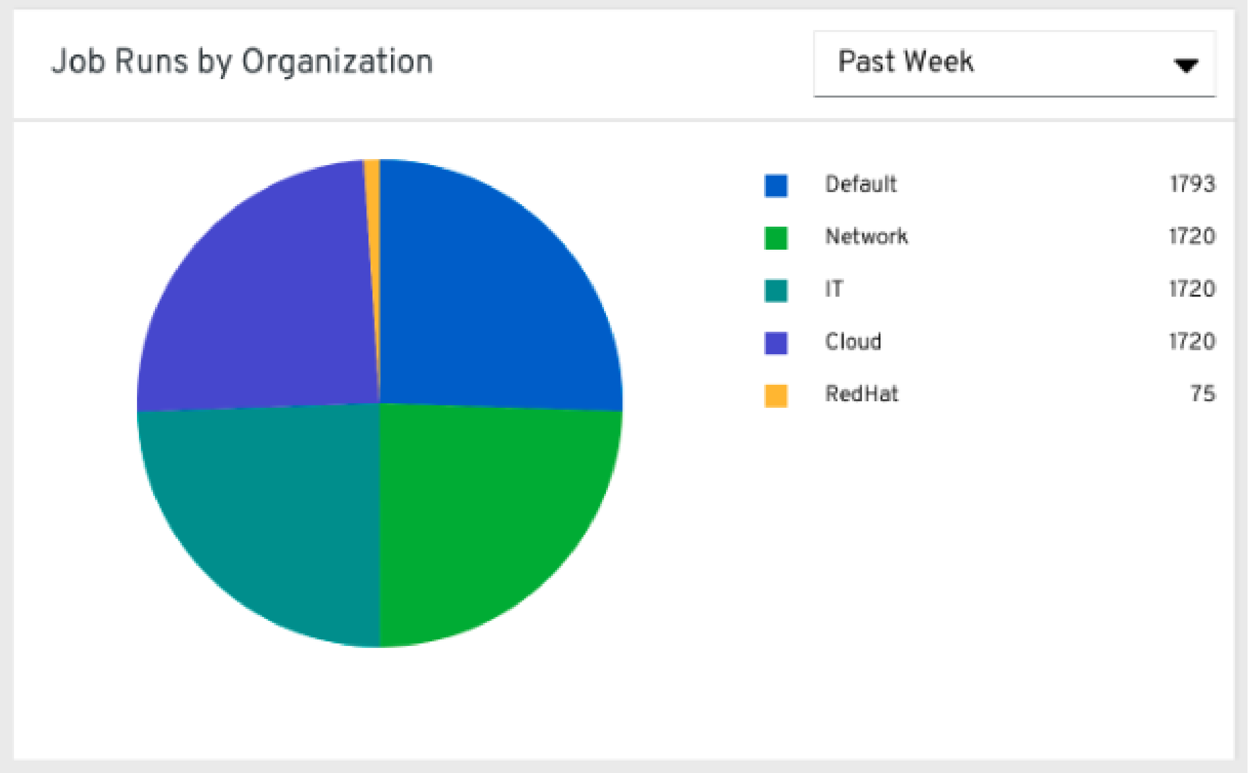
28.1.1 Usage by organization

This pie chart represents the number of tasks ran inside all jobs by a particular organization.



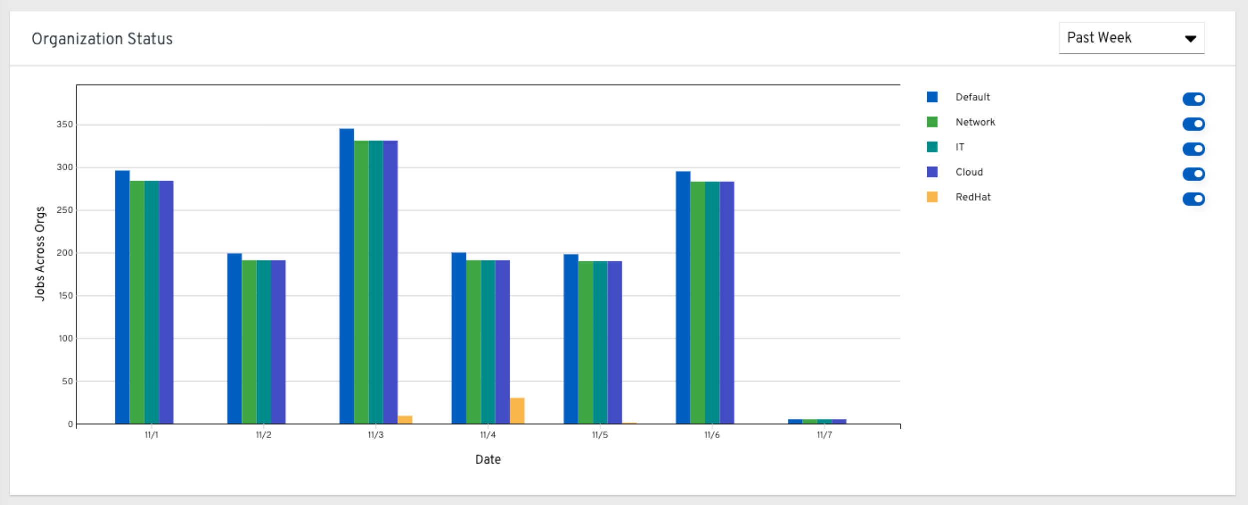
28.1.2 Job runs by organization

This pie chart represents Tower usage across *all* Tower clusters by organization, which is calculated by the number of jobs run by that organization.



28.1.3 Organization status

This bar chart represents Tower usage by organization and date, which is calculated by the number of jobs run by that organization on a particular date. Alternatively, you can specify to show the number of job runs per organization in one week, two weeks, and monthly increments.



28.2 Details of data collection

Automation Analytics collects certain classes of data from Ansible Tower:

- Basic configuration, like which features are enabled, and what operating system is being used
- Topology and status of the Tower environment and hosts, including capacity and health
- Counts of automation resources:
  - organizations, teams, and users
  - inventories and hosts
  - credentials (indexed by type)
  - projects (indexed by type)
  - templates
  - schedules
  - active sessions
  - running and pending jobs
- Job execution details (start time, finish time, launch type, and success)
- Automation task details (success, host id, playbook/role, task name, and module used)

You can use `awx-manage gather_analytics` (without `--ship`) to inspect the data that Tower sends so you can satisfy your data collection concerns. This will create a tarball that contains the analytics data that would be sent to Red Hat.

This file contains a number of JSON and CSV files. Each file contains a different set of analytics data.

- *events_table.csv*

28.2.1 manifest.json

manifest.json is the manifest of the analytics data. It describes each file included in the collection, and what version of the schema for that file is included. An example manifest is:

```
{
  "config.json": "1.1",
  "counts.json": "1.0",
  "cred_type_counts.json": "1.0",
  "events_table.csv": "1.1",
  "instance_info.json": "1.0",
  "inventory_counts.json": "1.2",
  "job_counts.json": "1.0",
  "job_instance_counts.json": "1.0",
  "org_counts.json": "1.0",
  "projects_by_scm_type.json": "1.0",
  "query_info.json": "1.0",
  "unified_job_template_table.csv": "1.0",
  "unified_jobs_table.csv": "1.0",
  "workflow_job_node_table.csv": "1.0",
  "workflow_job_template_node_table.csv": "1.0"
}
```

28.2.2 config.json

The config.json file contains a subset of the configuration endpoint `/api/v2/config` from the cluster. An example config.json is:

```
{
    "ansible_version": "2.9.1",
    "authentication_backends": [
        "social_core.backends.azuread.AzureADOAuth2",
        "django.contrib.auth.backends.ModelBackend"
    ],
    "external_logger_enabled": true,
    "external_logger_type": "splunk",
    "free_instances": 1234,
    "install_uuid": "d3d497f7-9d07-43ab-b8de-9d5cc9752b7c",
    "instance_uuid": "bed08c6b-19cc-4a49-bc9e-82c33936e91b",
    "license_expiry": 34937373,
    "license_type": "enterprise",
    "logging_aggregators": [
        "awx",
        "activity_stream",
        "job_events",
        "system_tracking"
    ],
    "pendo_tracking": "detailed",
    "platform": {
        "dist": [
            "redhat",
            "7.4",
```

(continues on next page)

(continued from previous page)

```
            "Maipo"
        ],
        "release": "3.10.0-693.el7.x86_64",
        "system": "Linux",
        "type": "traditional"
    },
    "total_licensed_instances": 2500,
    "tower_url_base": "https://ansible.rhdemo.io",
    "tower_version": "3.6.3"
}
```

A reference of fields collected:

ansible_version The system Ansible version on the host

authentication_backends What user authentication backends are available. See *Setting up Social Authentication* and *Setting up LDAP Authentication* for details

external_logger_enabled Whether external logging is enaled

external_logger_type What logging backend is in use if enabled. See *Tower Logging and Aggregation* for details

logging_aggregators What logging categories are sent to external logging. See *Tower Logging and Aggregation* for details

free_instances How many hosts are available in the license. A value of zero means the cluster is fully consuming its license.

install_uuid A UUID for the installation (identical for all cluster nodes)

instance_uuid A UUID for the instance (different for each cluster node)

license_expiry Time to expiry of the license, in seconds

license_type Type of the license (should be 'enterprise' for most cases)

pendo_tracking State of *Usability Analytics and Data Collection*

platform The operating system the cluster is running on

total_licensed_instances The total number of hosts in the license

tower_url_base The base URL for the cluster used by clients (shown in Automation Analytics)

tower_version Version of the software on the cluster

28.2.3 instance_info.json

The instance_info.json file contains detailed information on the instances that make up the cluster, organized by instance UUID. An example instance_info.json is:

```
{
    "bed08c6b-19cc-4a49-bc9e-82c33936e91b": {
        "capacity": 57,
        "cpu": 2,
        "enabled": true,
        "last_isolated_check": "2019-08-15T14:48:58.553005+00:00",
        "managed_by_policy": true,
        "memory": 8201400320,
        "uuid": "bed08c6b-19cc-4a49-bc9e-82c33936e91b",
```

(continues on next page)

(continued from previous page)

```
        "version": "3.6.3"
    }
    "c0a2a215-0e33-419a-92f5-e3a0f59bfaee": {
        "capacity": 57,
        "cpu": 2,
        "enabled": true,
        "last_isolated_check": "2019-08-15T14:48:58.553005+00:00",
        "managed_by_policy": true,
        "memory": 8201400320,
        "uuid": "c0a2a215-0e33-419a-92f5-e3a0f59bfaee",
        "version": "3.6.3"
    }
}
```

A reference of fields collected:

capacity The capacity of the instance for executing tasks. See <link> for details on how this is calculated.

cpu CPU cores for the instance

memory Memory for the instance

enabled Whether the instance is enabled and accepting tasks

managed_by_policy Whether the instance's membership in instance groups is managed by policy, or manually managed

version Version of the software on the instance

28.2.4 counts.json

The counts.json file contains the total number of objects for each relevant category in a cluster. An example counts.json is:

```
{
    "active_anonymous_sessions": 1,
    "active_host_count": 682,
    "active_sessions": 2,
    "active_user_sessions": 1,
    "credential": 38,
    "custom_inventory_script": 2,
    "custom_virtualenvs": 4,
    "host": 697,
    "inventories": {
        "normal": 20,
        "smart": 1
    },
    "inventory": 21,
    "job_template": 78,
    "notification_template": 5,
    "organization": 10,
    "pending_jobs": 0,
    "project": 20,
    "running_jobs": 0,
    "schedule": 16,
    "team": 5,
    "unified_job": 7073,
```

(continues on next page)

(continued from previous page)

```
    "user": 28,
    "workflow_job_template": 15
}
```

Each entry in this file is for the corresponding API objects in `/api/v2`, with the exception of the active session counts.

28.2.5 org_counts.json

The org_counts.json file contains information on each organization in the cluster, and the number of users and teams associated with that organization. An example org_counts.json is:

```
{
    "1": {
        "name": "Operations",
        "teams": 5,
        "users": 17
    },
    "2": {
        "name": "Development",
        "teams": 27,
        "users": 154
    },
    "3": {
        "name": "Networking",
        "teams": 3,
        "users": 28
    }
}
```

28.2.6 cred_type_counts.json

The cred_type_counts.json file contains information on the different credential types in the cluster, and how many credentials exist for each type. An example cred_type_counts.json is:

```
{
    "1": {
        "credential_count": 15,
        "managed_by_tower": true,
        "name": "Machine"
    },
    "2": {
        "credential_count": 2,
        "managed_by_tower": true,
        "name": "Source Control"
    },
    "3": {
        "credential_count": 3,
        "managed_by_tower": true,
        "name": "Vault"
    },
    "4": {
        "credential_count": 0,
```

(continues on next page)

(continued from previous page)

```
        "managed_by_tower": true,
        "name": "Network"
    },
    "5": {
        "credential_count": 6,
        "managed_by_tower": true,
        "name": "Amazon Web Services"
    },
    "6": {
        "credential_count": 0,
        "managed_by_tower": true,
        "name": "OpenStack"
    },
...
```

28.2.7 inventory_counts.json

The inventory_counts.json file contains information on the different inventories in the cluster. An example inventory_counts.json is:

```
{
    "1": {
        "hosts": 211,
        "kind": "",
        "name": "AWS Inventory",
        "source_list": [
            {
                "name": "AWS",
                "num_hosts": 211,
                "source": "ec2"
            }
        ],
        "sources": 1
    },
    "2": {
        "hosts": 15,
        "kind": "",
        "name": "Manual inventory",
        "source_list": [],
        "sources": 0
    },
    "3": {
        "hosts": 25,
        "kind": "",
        "name": "SCM inventory - test repo",
        "source_list": [
            {
                "name": "Git source",
                "num_hosts": 25,
                "source": "scm"
            }
        ],
        "sources": 1
    }
    "4": {
```

(continues on next page)

(continued from previous page)

```
        "num_hosts": 5,
        "kind": "smart",
        "name": "Filtered AWS inventory",
        "source_list": [],
        "sources": 0
    }
}
```

28.2.8 projects_by_scm_type.json

The projects_by_scm_type.json file provides a breakdown of all projects in the cluster, by source control type. An example projects_by_scm_type.json is:

```
{
    "git": 27,
    "hg": 0,
    "insights": 1,
    "manual": 0,
    "svn": 0
}
```

28.2.9 query_info.json

The query_info.json file provides details on when and how the data collection happened. An example query_info.json is:

```
{
    "collection_type": "manual",
    "current_time": "2019-11-22 20:10:27.751267+00:00",
    "last_run": "2019-11-22 20:03:40.361225+00:00"
}
```

collection_type is one of “manual” or “automatic”.

28.2.10 job_counts.json

The job_counts.json file provides details on the job history of the cluster, describing both how jobs were launched, and what their finishing status is. An example job_counts.json is:

```
{
    "launch_type": {
        "dependency": 3628,
        "manual": 799,
        "relaunch": 6,
        "scheduled": 1286,
        "scm": 6,
        "workflow": 1348
    },
    "status": {
        "canceled": 7,
        "failed": 108,
        "successful": 6958
```

(continues on next page)

(continued from previous page)

```
    },
    "total_jobs": 7073
}
```

28.2.11 job_instance_counts.json

The job_instance_counts.json file provides the same detail as job_counts.json, broken down by instance. An example job_instance_counts.json is:

```
{
    "localhost": {
        "launch_type": {
            "dependency": 3628,
            "manual": 770,
            "relaunch": 3,
            "scheduled": 1009,
            "scm": 6,
            "workflow": 1336
        },
        "status": {
            "canceled": 2,
            "failed": 60,
            "successful": 6690
        }
    }
}
```

Note that instances in this file are by hostname, not by UUID as they are in instance_info.

28.2.12 unified_job_template_table.csv

The unified_job_template_table.csv file provides information on job templates in the system. Each line contains the following fields for the job template:

id Job template id

name Job template name

polymorphic_ctype_id The id of the type of template it is

model The name of the polymorphic_ctype_id for the template. Examples include 'project', 'systemjobtemplate', 'jobtemplate', 'inventorysource', and 'workflowjobtemplate'

created When the template was created

modified When the template was last updated

created_by_id The userid that created the template. Blank if done by the system.

modified_by_id The userid that last modified the template. Blank if done by the system.

current_job_id Currently executing job id for the template, if any

last_job_id Last execution of the job

last_job_run Time of last execution of the job

last_job_failed Whether the last_job_id failed

status Status of last_job_id

next_job_run Next scheduled execution of the template, if any

next_schedule_id Schedule id for next_job_run, if any

28.2.13 unified_jobs_table.csv

The unified_jobs_table.csv file provides information on jobs run by the system. Each line contains the following fields for a job:

id Job id

name Job name (from the template)

polymorphic_ctype_id The id of the type of job it is

model The name of the polymorphic_ctype_id for the job. Examples include 'job', 'worfklow', and more.

organization_id The organization ID for the job

organization_name Name for the organization_id

created When the job record was created

started When the job started executing

finished When the job finished

elapsed Elapsed time for the job in seconds

unified_job_template_id The template for this job

launch_type One of "manual", "scheduled", "relaunched", "scm", "workflow", or "dependnecy"

schedule_id The id of the schedule that launched the job, if any

instance_group_id The instance group that executed the job

execution_node The node that executed the job (hostname, not UUID)

controller_node The controller node for the job, if run as an isolated job, or in a container group

cancel_flag Whether the job was cancelled

status Status of the job

failed Whether the job failed

job_explanation Any additional detail for jobs that failed to execute properly

28.2.14 workflow_job_template_node_table.csv

The workflow_job_template_node_table.csv provides information on the nodes defined in workflow job templates on the system.

Each line contains the following fields for a worfklow job template node:

id Node id

created When the node was created

modified When the node was last updated

unified_job_template_id The id of the job template, project, inventory, or other parent resource for this node

workflow_job_template_id The workflow job template that contains this node

inventory_id The inventory used by this node

success_nodes Nodes that are triggered after this node succeeds

failure_nodes Nodes that are triggered after this node fails

always_nodes Nodes that always are triggered after this node finishes

all_parents_must_converge Whether this node requires all its parent conditions satisfied to start

28.2.15 workflow_job_node_table.csv

The workflow_job_node_table.csv provides information on the jobs that have been executed as part of a workflow on the system.

Each line contains the following fields for a job run as part of a workflow:

id Node id

created When the node record was created

modified When the node record was last updated

job_id The job id for the job run for this node

unified_job_tempalte_id The id of the job template, project, inventory, or other parent resource for this job run

workflow_job_id The parent workflow job for this job run

inventory_id The inventory used by this job

success_nodes Nodes that were/would be triggered after this node succeded

failure_nodes Nodes that were/would be triggered after this node failed

always_nodes Nodes that were/would be triggered after this node finished

do_not_run Nodes that were not run in the workflow due to their start conditions not being triggered

all_parents_must_converge Whether this node required all its parent conditions satisfied to start

28.2.16 events_table.csv

The events_table.csv file provides information on all job events from all job runs in the system. Each line contains the following fields for a job event:

id Event id

uuid Event UUID

created When the event was created

parent_uuid The parent UUID for this event, if any

event The Ansible event type (such as runner_on_failed

task_action The module associated with this event, if any (such as 'command' or 'yum')

failed Whether the event returned "failed"

changed Whether the event returned "changed"

playbook Playbook associated with the event

play Play name from playbook

task Task name from playbook

role Role name from playbook

job_id Id of the job this event is from

host_id Id of the host this event is associated with, if any

host_name Name of the host this event is associated with, if any

start Start time of the task

end End time of the task

duration Duration of the task

warnings Any warnings from the task/module

deprecations Any deprecation warnings from the task/module

CHAPTER

TWENTYNINE

TROUBLESHOOTING TOWER

29.1 Error logs

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

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

29.2 sosreport

The `sosreport` is a utility that collects diagnostic information for Support to be able to use to analyze and investigate the issues you report. To properly provide Technical Support this information, refer to the Knowledgebase article for sosreport from the Red Hat Customer portal to perform the following procedures:

1. Install the sosreport utility.
2. Generate an sosreport.
3. Provide the sosreport to Red Hat Support.

29.3 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.)

29.4 Unable to login to Tower via HTTP

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

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

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

To apply the changes, run:

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

29.5 WebSockets port for live events not working

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

29.6 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.

29.7 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`.

29.8 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/
```

29.9 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.

29.10 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.

29.11 Reusing an external database causes installations to fail

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

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

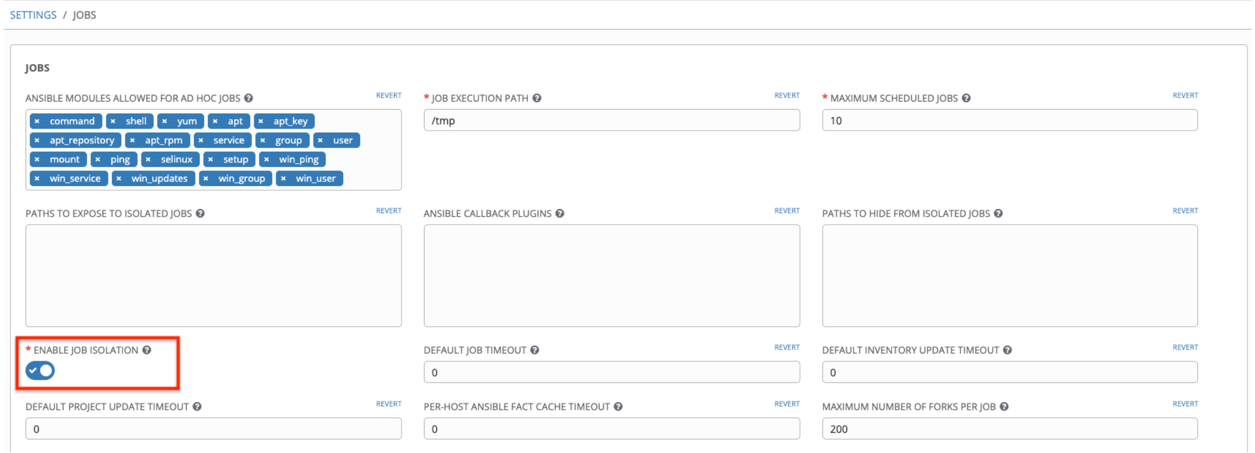
When setting up an external database which has been used in a prior installation, the database used for the clustered node must be manually cleared before any additional installations can succeed.

29.11.1 Bubblewrap functionality and variables

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

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

1. Click the Settings (⚙) icon from the left navigation bar.
2. Click the **Jobs** tab.
3. In the **Enable Job Isolation**, change the toggle button selection to **OFF** to disable bubblewrap support or select **ON** to enable it.



SETTINGS / JOBS

JOBS

ANSIBLE MODULES ALLOWED FOR AD HOC JOBS: command, shell, yum, apt, apt_key, apt_repository, apt_rpm, service, group, user, mount, ping, selinux, setup, win_ping, win_service, win_updates, win_group, win_user

* JOB EXECUTION PATH: /tmp

* MAXIMUM SCHEDULED JOBS: 10

PATHS TO EXPOSE TO ISOLATED JOBS

ANSIBLE CALLBACK PLUGINS

PATHS TO HIDE FROM ISOLATED JOBS

* ENABLE JOB ISOLATION: ON

DEFAULT JOB TIMEOUT: 0

DEFAULT INVENTORY UPDATE TIMEOUT: 0

DEFAULT PROJECT UPDATE TIMEOUT: 0

PER-HOST ANSIBLE FACT CACHE TIMEOUT: 0

MAXIMUM NUMBER OF FORKS PER JOB: 200

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

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

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

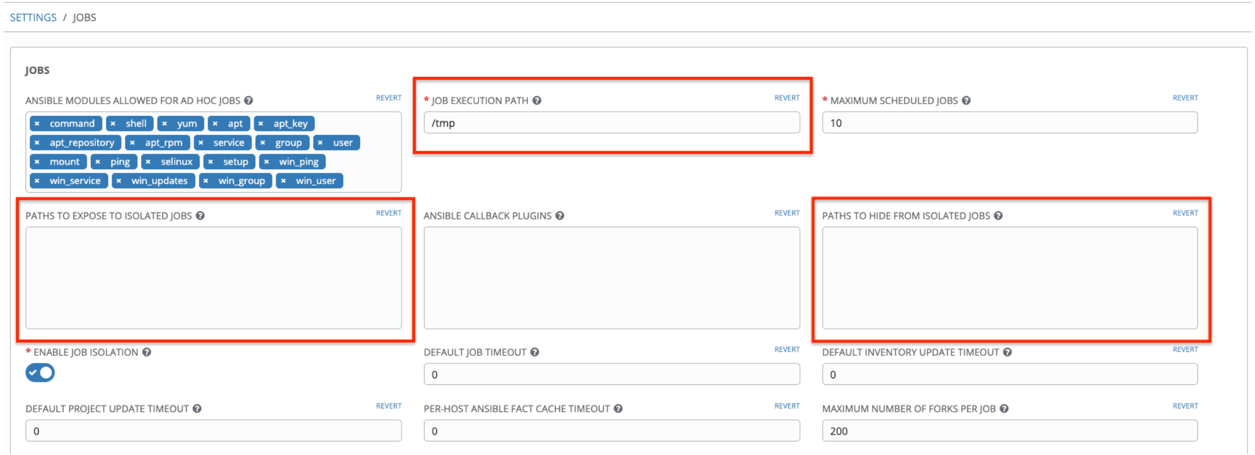
```
AWX_PROOT_HIDE_PATHS = ['/list/of/', '/paths']
```

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

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

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

The above fields can be found in the Jobs Settings window:



SETTINGS / JOBS

JOBS

ANSIBLE MODULES ALLOWED FOR AD HOC JOBS: command, shell, yum, apt, apt_key, apt_repository, apt_rpm, service, group, user, mount, ping, selinux, setup, win_ping, win_service, win_updates, win_group, win_user

* JOB EXECUTION PATH: /tmp

* MAXIMUM SCHEDULED JOBS: 10

PATHS TO EXPOSE TO ISOLATED JOBS:

ANSIBLE CALLBACK PLUGINS:

PATHS TO HIDE FROM ISOLATED JOBS:

* ENABLE JOB ISOLATION: ● on

DEFAULT JOB TIMEOUT: 0

DEFAULT INVENTORY UPDATE TIMEOUT: 0

DEFAULT PROJECT UPDATE TIMEOUT: 0

PER-HOST ANSIBLE FACT CACHE TIMEOUT: 0

MAXIMUM NUMBER OF FORKS PER JOB: 200

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

29.12 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.

29.13 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

THIRTY

TOWER TIPS AND TRICKS

30.1 Using the Tower CLI Tool

Ansible Tower has a full-featured command line interface. Refer to AWX CLI Ansible Tower documentation for configuration and usage instructions.

30.2 Changing the Tower Admin Password

During the installation process, you are prompted to enter an administrator password which is used for the `admin` superuser/first user created in Tower. If you log into the instance via SSH, it will tell you the default admin password in the prompt. If you need to change this password at any point, run the following command as root on the Tower server:

```
awx-manage changepassword admin
```

Next, enter a new password. After that, the password you have entered will work as the admin password in the web UI.

To set policies at creation time for password validation using Django, see *Django password policies* for detail.

30.3 Creating a Tower Admin from the commandline

Once in a while you may find it helpful to create an admin (superuser) account from the commandline. To create an admin, run the following command as root on the Tower server and enter in the admin information as prompted:

```
awx-manage createsuperuser
```

30.4 Setting up a jump host to use with Tower

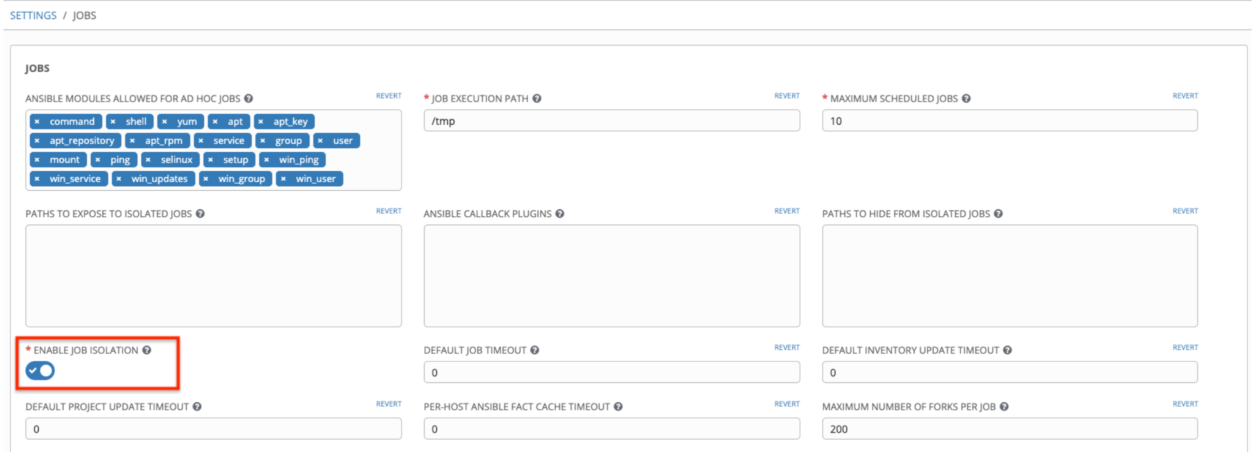
Credentials supplied by Tower will not flow to the jump host via ProxyCommand. They are only used for the end-node once the tunneled connection is set up.

To make this work, configure a fixed user/keyfile in the AWX user's SSH config in the ProxyCommand definition that sets up the connection through the jump host. For example:

```
Host tampa
Hostname 10.100.100.11
IdentityFile [privatekeyfile]

Host 10.100..
Proxycommand ssh -W [jumphostuser]@%h:%p tampa
```

Note: You must disable PRoot by default if you need to use a jump host. You can disable PRoot through the Configure Tower user interface by setting the **Enable Job Isolation** toggle to **OFF** from the Jobs tab:



SETTINGS / JOBS

JOBS

ANSIBLE MODULES ALLOWED FOR AD HOC JOBS: command, shell, yum, apt, apt_key, apt_repository, apt_rpm, service, group, user, mount, ping, selinux, setup, win_ping, win_service, win_updates, win_group, win_user

* JOB EXECUTION PATH: /tmp

* MAXIMUM SCHEDULED JOBS: 10

PATHS TO EXPOSE TO ISOLATED JOBS

ANSIBLE CALLBACK PLUGINS

PATHS TO HIDE FROM ISOLATED JOBS

* ENABLE JOB ISOLATION: ✓

DEFAULT JOB TIMEOUT: 0

DEFAULT INVENTORY UPDATE TIMEOUT: 0

DEFAULT PROJECT UPDATE TIMEOUT: 0

PER-HOST ANSIBLE FACT CACHE TIMEOUT: 0

MAXIMUM NUMBER OF FORKS PER JOB: 200

You can also add a jump host to your Tower instance through Inventory variables. These variables can be set at either the inventory, group, or host level. To add this, navigate to your inventory and in the `variables` field of whichever level you choose, add the following variables:

```
ansible_user: <user_name>
ansible_connection: ssh
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q <user_name>@<jump_server_
↪name>"'
```

30.5 View Ansible outputs for JSON commands when using Tower

When working with Ansible Tower, you can use the API to obtain the Ansible outputs for commands in JSON format.

To view the Ansible outputs, browse to:

```
https://<tower server name>/api/v2/jobs/<job_id>/job_events/
```

30.6 Locate and configure the Ansible configuration file

While Ansible does not require a configuration file, OS packages often include a default one in `/etc/ansible/ansible.cfg` for possible customization. In order to use a custom `ansible.cfg` file, place it at the root of your project. Ansible Tower runs `ansible-playbook` from the root of the project directory, where it will then find the custom `ansible.cfg` file. An `ansible.cfg` anywhere else in the project will be ignored.

To learn which values you can use in this file, refer to the configuration file on github.

Using the defaults are acceptable for starting out, but know that you can configure the default module path or connection type here, as well as other things.

Tower overrides some ansible.cfg options. For example, Tower stores the SSH ControlMaster sockets, the SSH agent socket, and any other per-job run items in a per-job temporary directory, secured by multi-tenancy access control restrictions via PRoot.

30.7 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

30.8 The ALLOW_JINJA_IN_EXTRA_VARS variable

Setting `ALLOW_JINJA_IN_EXTRA_VARS = template` only works for saved job template extra variables. Prompted variables and survey variables are excluded from the 'template'. This parameter has three values: `template` to allow usage of Jinja saved directly on a job template definition (the default), `never` to disable all Jinja usage (recommended), and `always` to always allow Jinja (strongly discouraged, but an option for prior compatibility).

30.9 Using virtualenv with Ansible Tower

Virtualenv creates isolated Python environments to avoid problems caused by conflicting dependencies and differing versions. Virtualenv works by simply creating a folder which contains all of the necessary executables and dependencies for a specific version of Python. Ansible Tower creates two virtualenvs during installation–one is used to run Tower, while the other is used to run Ansible. This allows Tower to run in a stable environment, while allowing you to add or update modules to your Ansible Python environment as necessary to run your playbooks. For more information on virtualenv, see the Python Guide to Virtual Environments and the *Python virtualenv* project itself.

By default, the virtualenv is located at `/var/lib/awx/venv/ansible` on the file system but you can create your own custom directories and use them in inventory imports. This allows you to choose how you run your inventory imports, as inventory sources use custom virtual environments.

Tower also pre-installs a variety of third-party library/SDK support into this virtualenv for its integration points with a variety of cloud providers (such as EC2, OpenStack, Azure, etc.) Periodically, you may want to add additional SDK support into this virtualenv, which is described in further detail below.

Note: It is highly recommended that you run `umask` 0022 before installing any packages to the virtual environment. Failure to properly configure permissions can result in Tower service failures. An example follows:

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

In addition to adding modules to the virtualenv that Tower uses to run Ansible, you can create new virtualenvs as described below.

30.9.1 Preparing a new custom virtualenv

You can specify a different virtualenv for running Job Templates in Tower. In order to do so, you must specify which directories those venvs reside. You could choose to keep custom venvs inside `/var/lib/awx/venv/`, but it is highly recommended that a custom directory be created. The following examples use a placeholder directory `/opt/my-envs/`, but you can replace this with a directory path of your choice anywhere this is specified.

1. Preparing a new custom virtualenv requires the virtualenv package to be pre-installed:

```
$ sudo yum install python-virtualenv
```

2. Create a directory for your custom venvs:

```
$ sudo mkdir /opt/my-envs
```

3. Make sure to give your directory the appropriate write permission, execution permission and ownership:

```
$ sudo chmod 0755 /opt/my-envs
$ sudo chown awx:awx /opt/my-envs
```

4. Optionally, you can specify in Tower which directory to look for custom venvs by adding this directory to the `CUSTOM_VENV_PATHS` setting as follows:

```
$ curl -X PATCH 'https://user:password@tower.example.org/api/v2/settings/system/' \
    -d '{"CUSTOM_VENV_PATHS": ["/opt/my-envs/"]}'  -H 'Content-Type:application/json'
```

If you have venvs spanned over multiple directories, add all the paths and Tower will aggregate venvs from them:

```
$ curl -X PATCH 'https://user:password@tower.example.org/api/v2/settings/system/' \
    -d '{"CUSTOM_VENV_PATHS": ["/path/1/to/venv/", "/path/2/to/venv/", "/path/3/to/
↪venv/"]}' \
    -H 'Content-Type:application/json'
```

5. Now that a venv directory has been set up, create a virtual environment in that location:

```
$ sudo virtualenv /opt/my-envs/custom-venv
```

Note: Multiple versions of Python are supported, but the syntax for creating virtualenvs in Python 3 has changed slightly: `$ sudo python3 -m venv /opt/my-envs/custom-venv`

6. Next, install gcc so that `psutil` can be compiled:

```
$ yum install gcc
```

7. Python header files are needed to compile `psutil`. The package needed to successfully compile psutil on RHEL 8 systems is `platform-python-devel`:

```
$ yum install platform-python-devel
```

8. Your newly created virtualenv needs a few base dependencies to properly run playbooks (eg., fact gathering):

```
$ sudo /opt/my-envs/custom-venv/bin/pip install psutil
```

From here, you can install *additional* Python dependencies that you care about, such as a per-virtualenv version of Ansible itself:

```
$ sudo /opt/my-envs/custom-venv/bin/pip install -U "ansible == X.Y.Z"
```

Or you can add an additional third-party SDK that is not included with the base Tower installation:

```
$ sudo /opt/my-envs/custom-venv/bin/pip install -U python-digitalocean
```

If you want to copy them, the libraries included in Tower's default virtualenv can be found using `pip freeze`:

```
$ sudo /var/lib/awx/venv/ansible/bin/pip freeze
```

In a clustered Tower installation, you need to ensure that the same custom virtualenv exists on **every** local file system at `/opt/my-envs/`. Custom virtualenvs are supported on isolated instances. If you are using a custom virtual environment, it needs to also be copied or replicated on any isolated node you would be using, not just on the Tower node. For setting up custom virtual environments in containers, refer to the *OpenShift Deployment and Configuration* section of the *Ansible Tower Administration Guide*.

30.9.2 Assigning custom virtualenvs

Once you have created a custom virtualenv, you can assign it at the Organization, Project, or Job Template level to use it in job runs. You can set the custom venv on an inventory source to run inventory updates in that venv. Jobs using that inventory follow their own rules and will not use this venv. If an SCM inventory source does not have a venv selected, it can use the venv of its linked project. You can assign a custom venv on the organization, but if you do, it will not be used by inventory updates in the organization, as it is only used in job runs.

The following shows the proper way to assign a custom venv at the desired level.

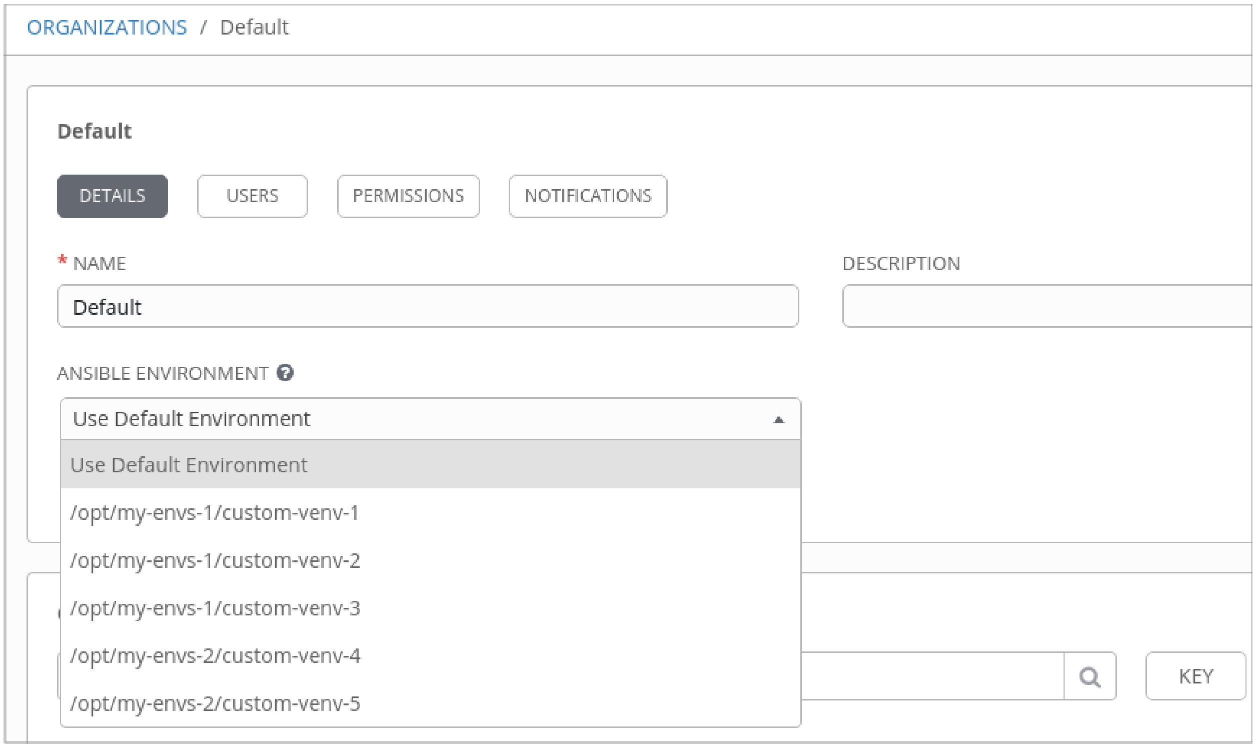
```
PATCH https://awx-host.example.org/api/v2/organizations/N/
PATCH https://awx-host.example.org/api/v2/projects/N/
PATCH https://awx-host.example.org/api/v2/job_templates/N/
PATCH https://awx-host.example.org/api/v2/inventory_sources/N/

Content-Type: application/json
{
        'custom_virtualenv': '/opt/my-envs/custom-venv'
}
```

An HTTP GET request to `/api/v2/config/` provides a list of detected installed virtualenvs:

```
{
        "custom_virtualenvs": [
                "/opt/my-envs/custom-venv",
                "/opt/my-envs/my-other-custom-venv",
        ],
...
}
```

You can also specify the virtual environment to assign to an Organization, Project, and Job Template from their respective edit screens in the Ansible Tower User Interface. Select the virtualenv from the **Ansible Environment** drop-down menu, as shown in the example below:



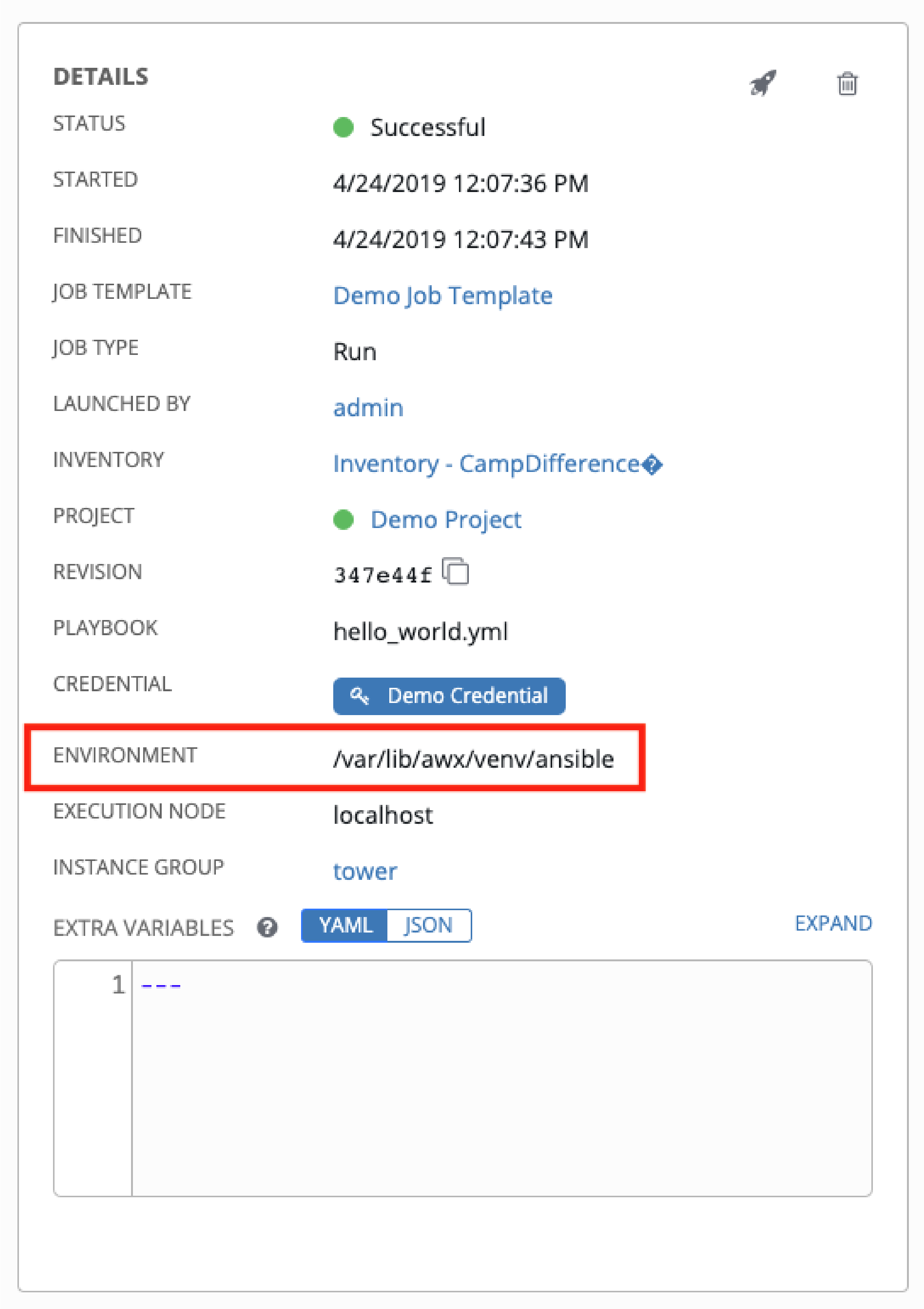
* NAME: Default

DESCRIPTION:

ANSIBLE ENVIRONMENT: Use Default Environment

- Use Default Environment
- /opt/my-envs-1/custom-venv-1
- /opt/my-envs-1/custom-venv-2
- /opt/my-envs-1/custom-venv-3
- /opt/my-envs-2/custom-venv-4
- /opt/my-envs-2/custom-venv-5

When you launch a job template, you will also see the virtualenv specified in the Job Details pane:



DETAILS

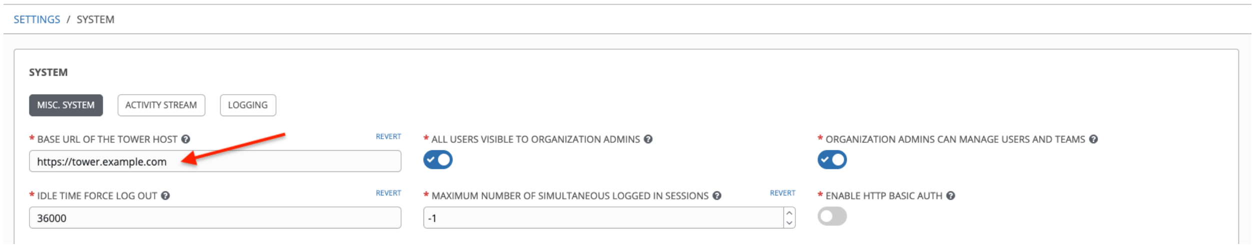
STATUS	● Successful
STARTED	4/24/2019 12:07:36 PM
FINISHED	4/24/2019 12:07:43 PM
JOB TEMPLATE	Demo Job Template
JOB TYPE	Run
LAUNCHED BY	admin
INVENTORY	Inventory - CampDifference�
PROJECT	● Demo Project
REVISION	347e44f
PLAYBOOK	hello_world.yml
CREDENTIAL	Demo Credential
ENVIRONMENT	/var/lib/awx/venv/ansible
EXECUTION NODE	localhost
INSTANCE GROUP	tower

EXTRA VARIABLES

```
---
```

30.10 Configuring the `towerhost` hostname for notifications

In the *System Settings*, you can replace `https://tower.example.com` in the **Base URL of The Tower Host** field with your preferred hostname to change the notification hostname.



* BASE URL OF THE TOWER HOST: https://tower.example.com

* ALL USERS VISIBLE TO ORGANIZATION ADMINS: ✓ (on)

* ORGANIZATION ADMINS CAN MANAGE USERS AND TEAMS: ✓ (on)

* IDLE TIME FORCE LOG OUT: 36000

* MAXIMUM NUMBER OF SIMULTANEOUS LOGGED IN SESSIONS: -1

* ENABLE HTTP BASIC AUTH: ○

Refreshing your Tower license also changes the notification hostname. New installations of Ansible Tower should not have to set the hostname for notifications.

30.11 Launching Jobs with curl

Launching jobs with the Tower API is simple. Here are some easy to follow examples using the `curl` tool.

Assuming that your Job Template ID is '1', your Tower IP is 192.168.42.100, and that `admin` and `awxsecret` are valid login credentials, you can create a new job this way:

```
curl -f -k -H 'Content-Type: application/json' -XPOST \
    --user admin:awxsecret \
    http://192.168.42.100/api/v2/job_templates/1/launch/
```

This returns a JSON object that you can parse and use to extract the 'id' field, which is the ID of the newly created job.

You can also pass extra variables to the Job Template call, such as is shown in the following example:

```
curl -f -k -H 'Content-Type: application/json' -XPOST \
    -d '{"extra_vars": "{\"foo\": \"bar\"}"}' \
    --user admin:awxsecret http://192.168.42.100/api/v2/job_templates/1/launch/
```

You can view the live API documentation by logging into http://192.168.42.100/api/ and browsing around to the various objects available.

Note: The `extra_vars` parameter needs to be a string which contains JSON, not just a JSON dictionary, as you might expect. Use caution when escaping the quotes, etc.

30.12 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.

30.13 Filtering instances returned by the dynamic inventory sources in Tower

By default, the dynamic inventory sources in Tower (AWS, Google, 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.

30.14 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.

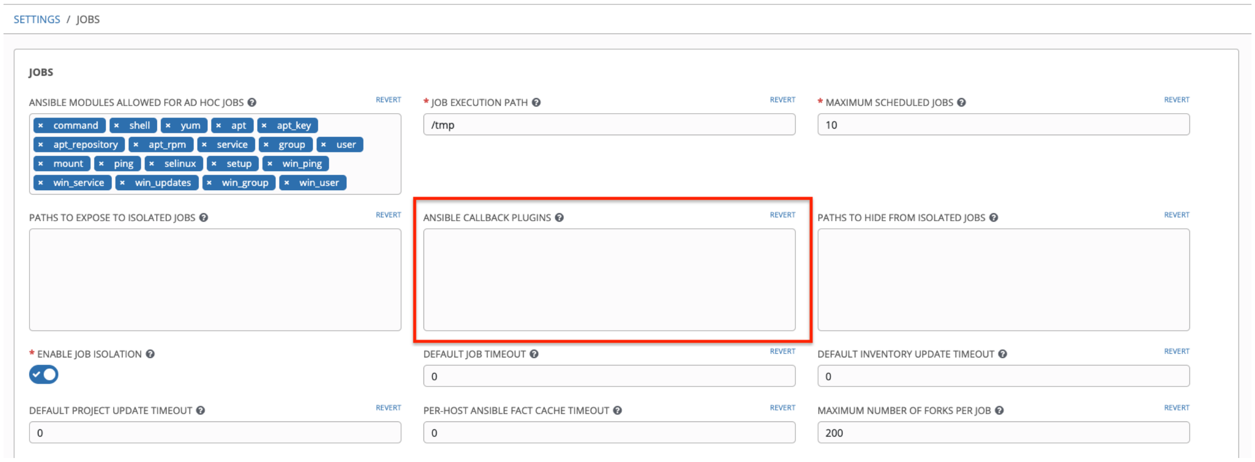
30.15 Using callback plugins with Tower

Ansible has a flexible method of handling actions during playbook runs, called callback plugins. You can use these plugins with Tower to do things like notify services upon playbook runs or failures, send emails after every playbook run, etc. For official documentation on the callback plugin architecture, refer to: http://docs.ansible.com/developing_plugins.html#callbacks

Note: Ansible Tower does not support the `stdout` callback plugin because Ansible only allows one, and it is already being used by Ansible Tower for streaming event data.

You may also want to review some example plugins, which should be modified for site-specific purposes, such as those available at: https://github.com/ansible/ansible/tree/devel/lib/ansible/plugins/callback

To use these plugins, put the callback plugin `.py` file into a directory called `/callback_plugins` alongside your playbook in your Tower Project. Then, specify their paths (one path per line) in the **Ansible Callback Plugins** field of the Configure Tower Job settings screen:



JOBS

ANSIBLE MODULES ALLOWED FOR AD HOC JOBS: command, shell, yum, apt, apt_key, apt_repository, apt_rpm, service, group, user, mount, ping, selinux, setup, win_ping, win_service, win_updates, win_group, win_user

* JOB EXECUTION PATH: /tmp

* MAXIMUM SCHEDULED JOBS: 10

PATHS TO EXPOSE TO ISOLATED JOBS

ANSIBLE CALLBACK PLUGINS

PATHS TO HIDE FROM ISOLATED JOBS

* ENABLE JOB ISOLATION: ✓

DEFAULT JOB TIMEOUT: 0

DEFAULT INVENTORY UPDATE TIMEOUT: 0

DEFAULT PROJECT UPDATE TIMEOUT: 0

PER-HOST ANSIBLE FACT CACHE TIMEOUT: 0

MAXIMUM NUMBER OF FORKS PER JOB: 200

Note: To have most callbacks shipped with Ansible applied globally, you must add them to the `callback_whitelist` section of your `ansible.cfg`. If you have a custom callbacks, refer to the Ansible documentation for Enabling callback plugins.

30.16 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.

30.17 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
```

(continues on next page)

(continued from previous page)

```
|   `-- mygroup
|-- host_vars
|   `-- myhost
`-- hosts
```

To import these hosts and vars, run the `awx-manage` command:

```
awx-manage inventory_import --source=inventory/ \
  --inventory-name="My Tower Inventory"
```

If you only have a single flat file of inventory, a file called ansible-hosts, for example, import it like the following:

```
awx-manage inventory_import --source=./ansible-hosts \
  --inventory-name="My Tower Inventory"
```

In case of conflicts or to overwrite an inventory named “My Tower Inventory”, run:

```
awx-manage inventory_import --source=inventory/ \
  --inventory-name="My Tower Inventory" \
  --overwrite --overwrite-vars
```

If you receive an error, such as:

```
ValueError: need more than 1 value to unpack
```

Create a directory to hold the hosts file, as well as the group_vars:

```
mkdir -p inventory-directory/group_vars
```

Then, for each of the groups that have :vars listed, create a file called `inventory-directory/group_vars/<groupname>` and format the variables in YAML format.

Once broken out, the importer will handle the conversion correctly.

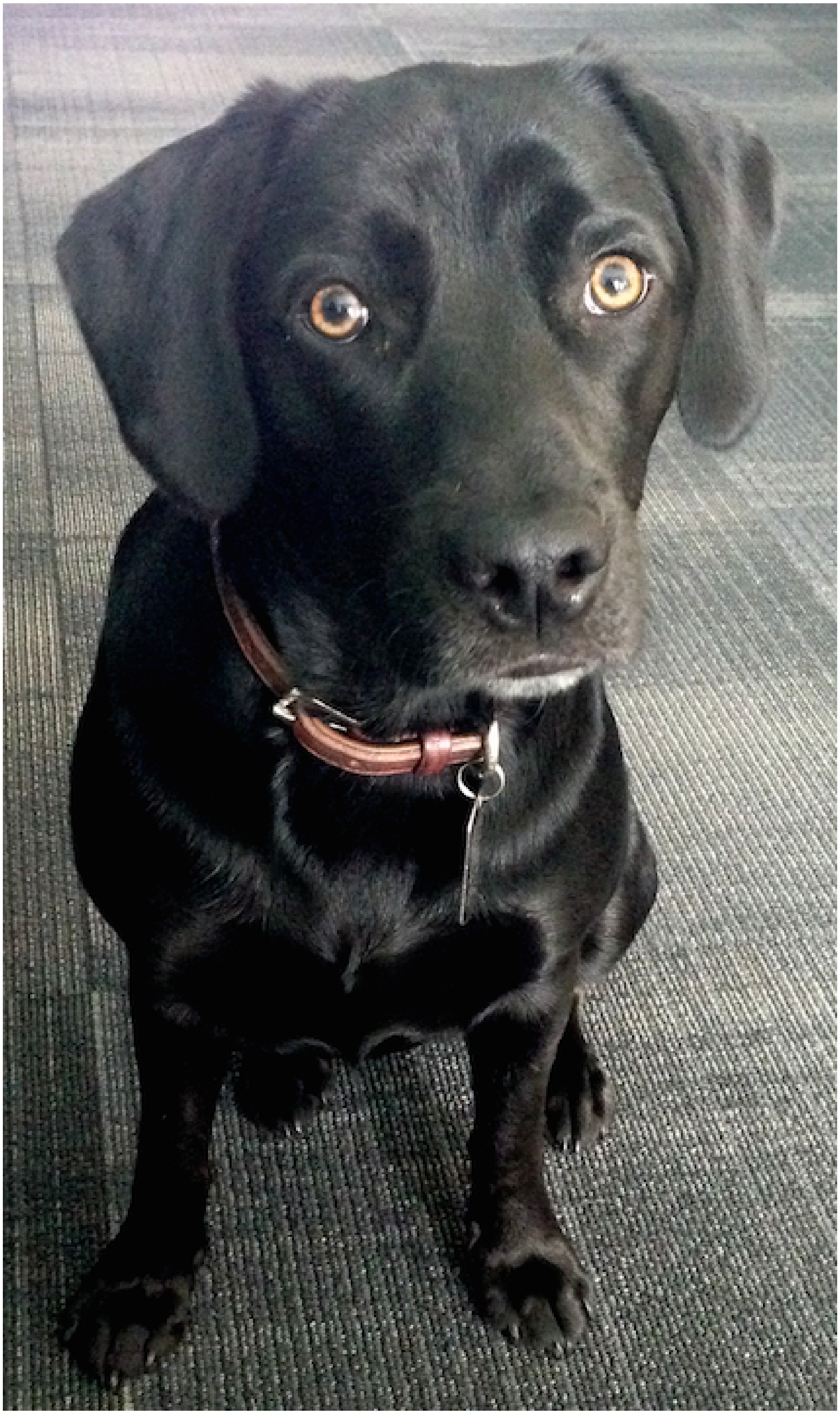
CHAPTER THIRTYONE

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

THIRTYTWO

INDEX

- genindex

CHAPTER THIRTYTHREE

COPYRIGHT © RED HAT, INC.

Third Party Rights

INDEX