Ansible Tower API Guide

Release Ansible Tower 3.0.1

Red Hat, Inc.

Jun 06, 2017

CONTENTS

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

The *Ansible Tower API Guide* focuses on helping you understand the Ansible Tower API. This document has been updated to include information for the latest release of Ansible Tower 3.0.1.

Ansible Tower Version 3.0.1; July 28, 2016; https://access.redhat.com/

ONE

TOOLS

This document offers a basic understanding of the REST API used by Ansible Tower.

REST stands for Representational State Transfer and is sometimes spelled as "ReST". It relies on a stateless, client-server, and cacheable communications protocol, usually the HTTP protocol.

You may find it helpful see which API calls Tower makes in sequence. To do this, you can use the UI from Firebug or Chrome with developer plugins.

Another alternative is Charles Proxy (http://www.charlesproxy.com/), which offers a visualizer that you may find helpful. While it is commercial software, it can insert itself as an OS X proxy, for example, and intercept both requests from web browsers as well as curl and other API consumers.

Other alternatives include:

- Fiddler (http://www.telerik.com/fiddler)

- mitmproxy (https://mitmproxy.org/)

- Live HTTP headers FireFox extension (https://addons.mozilla.org/en-US/firefox/addon/live-http-headers/)

- Paros (http://sourceforge.net/projects/paros/)

BROWSABLE API

REST APIs provide access to resources (data entities) via URI paths. You can visit the Ansible Tower REST API in a web browser at: `http://<Tower server name>/api/`

Clicking on various links in the API allows you to explore related resources.

```
HTTP 200 OK
Content-Type: application/json
Vary: Accept
Allow: GET, HEAD, OPTIONS
X-API-Time: 0.004s


{
    "authtoken": "/api/v1/authtoken/",
    "ping": "/api/v1/ping/",
    "config": "/api/v1/config/",
    "me": "/api/v1/me/",
    "dashboard": "/api/v1/dashboard/",
    "organizations": "/api/v1/organizations/",
    "users": "/api/v1/users/",
    "projects": "/api/v1/projects/",
    "teams": "/api/v1/teams/",
    "credentials": "/api/v1/credentials/",
    "inventory": "/api/v1/inventories/",
    "inventory_scripts": "/api/v1/inventory_scripts/",
    "inventory_sources": "/api/v1/inventory_sources/",
    "groups": "/api/v1/groups/",
    "hosts": "/api/v1/hosts/",
    "job_templates": "/api/v1/job_templates/",
    "jobs": "/api/v1/jobs/",
    "ad_hoc_commands": "/api/v1/ad_hoc_commands/",
    "system_job_templates": "/api/v1/system_job_templates/",
    "system_jobs": "/api/v1/system_jobs/",
    "schedules": "/api/v1/schedules/",
    "unified_job_templates": "/api/v1/unified_job_templates/",
    "unified_jobs": "/api/v1/unified_jobs/",
    "activity_stream": "/api/v1/activity_stream/"
}
```

Clicking on the next to the page name (toward the top of the screen) for an API endpoint gives you documentation on the access methods for that particular API endpoint and what data is returned when using those methods.

You can also use PUT and POST verbs on the specific API pages by formatting JSON in the various text fields.

THREE

CONVENTIONS

Tower uses a standard REST API, rooted at `/api/` on the server. The API is versioned for compatibility reasons, but only `/api/v1/` is currently available. You can see information about what API versions are available by querying `/api/`.

You may have to specify the content/type on POST or PUT requests accordingly.

- PUT: Update a specific resource (by an identifier) or a collection of resources. PUT can also be used to create a specific resource if the resource identifier is know before-hand.

- POST: Create a new resource. Also acts as a catch-all verb for operations that do not fit into the other categories.

All URIs not ending with `"/"` receive a 301 redirect.

Note: Ansible Tower 3.0.1 API change: Formatting of extra_vars attached to Job Template records is preserved. Previously, YAML would be converted to JSON and returned as JSON. In 2.2.0 and newer, YAML is returned as YAML with formatting and comments preserved, and JSON is returned as JSON.

SORTING

To provide examples that are easy to follow, the following URL is used throughout this guide:

```
http://<Tower server name>/api/v1/groups/
```

To specify that {{ model_verbose_name_plural }} are returned in a particular order, use the `order_by` query string parameter on the GET request.

```
http://<Tower server name>/api/v1/model_verbose_name_plural?order_by={{ order_field }}
```

Prefix the field name with a dash (−) to sort in reverse:

```
http://<Tower server name>/api/v1/model_verbose_name_plural?order_by=-{{ order_field }
↪}
```

Multiple sorting fields may be specified by separating the field names with a comma (,):

```
http://<Tower server name>/api/v1/model_verbose_name_plural?order_by={{ order_field }}
↪,some_other_field
```

SEARCHING

Use the search query string parameter to perform a case-insensitive search within all designated text fields of a model.

```
http://<Tower server name>/api/v1/model_verbose_name?search=findme
```

(Note: Added in AWX 1.4)

FILTERING

Any collection is what the system calls a "queryset" and can be filtered via various operators.

For example, to find the groups that contain the name "foo":

```
http://<Tower server name>/api/v1/groups/?name__contains=foo
```

To find an exact match:

```
http://<Tower server name>/api/v1/groups/?name=foo
```

If a resource is of an integer type, you must add $\_\_int$ to the end to cast your string input value to an integer, like so:

```
http://<Tower server name>/api/v1/arbitrary_resource/?x__int=5
```

Related resources can also be queried, like so:

```
http://<Tower server name>/api/v1/groups/?user__firstname__icontains=john
```

This will return all groups with users with names that include the string "John" in them.

You can also filter against multiple fields at once:

```
http://<Tower server name>/api/v1/groups/?user__firstname__icontains=john&group__name_
↪__icontains=foo
```

This finds all groups containing a user whose name contains "John" where the group contains the string foo.

For more about what types of operators are available, refer to:

> https://docs.djangoproject.com/en/dev/ref/models/querysets/

Note: You can also watch the API as the UI is being used to see how it is filtering on various criteria.

Any additional query string parameters may be used to filter the list of results returned to those matching a given value. Only fields and relations that exist in the database may be used for filtering. Any special characters in the specified value should be url-encoded. For example:

```
?field=value%20xyz
```

Fields may also span relations, only for fields and relationships defined in the database:

```
?other__field=value
```

To exclude results matching certain criteria, prefix the field parameter with `not__`:

```
?not__field=value
```

(Added in AWX 1.4) By default, all query string filters are AND'ed together, so only the results matching all filters will be returned. To combine results matching any one of multiple criteria, prefix each query string parameter with `or__`:

```
?or__field=value&or__field=othervalue
?or__not__field=value&or__field=othervalue
```

(Added in Ansible Tower 1.4.5) The default AND filtering applies all filters simultaneously to each related object being filtered across database relationships. The chain filter instead applies filters separately for each related object. To use, prefix the query string parameter with `chain__`:

```
?chain__related__field=value&chain__related__field2=othervalue
?chain__not__related__field=value&chain__related__field2=othervalue
```

If the first query above were written as `?related__field=value&related__field2=othervalue`, it would return only the primary objects where the same related object satisfied both conditions. As written using the chain filter, it would return the intersection of primary objects matching each condition.

Field lookups may also be used for more advanced queries, by appending the lookup to the field name:

```
?field__lookup=value
```

The following field lookups are supported:

- `exact`: Exact match (default lookup if not specified).
- `iexact`: Case-insensitive version of exact.
- `contains`: Field contains value.
- `icontains`: Case-insensitive version of contains.
- `startswith`: Field starts with value.
- `istartswith`: Case-insensitive version of startswith.
- `endswith`: Field ends with value.
- `iendswith`: Case-insensitive version of endswith.
- `regex`: Field matches the given regular expression.
- `iregex`: Case-insensitive version of regex.
- `gt`: Greater than comparison.
- `gte`: Greater than or equal to comparison.
- `lt`: Less than comparison.
- `lte`: Less than or equal to comparison.
- `isnull`: Check whether the given field or related object is null; expects a boolean value.
- `in`: Check whether the given field's value is present in the list provided; expects a list of items.
- Boolean values may be specified as `True` or `1` for true, `False` or `0` for false (both case-insensitive).

Null values may be specified as `None` or `Null` (both case-insensitive), though it is preferred to use the `isnull` lookup to explicitly check for null values.

Lists (for the `in` lookup) may be specified as a comma-separated list of values.

PAGINATION

Responses for collections in the API are paginated. This means that while a collection may contain tens or hundreds of thousands of objects, in each web request, only a limited number of results are returned for API performance reasons.

When you get back the result for a collection you will see something similar to the following:

```
{'count': 25, 'next': 'http://testserver/api/v1/some_resource?page=2', 'previous':
→None, 'results': [ ... ] }
```

To get the next page, simply request the page given by the 'next' sequential URL.

Use the `page_size=XX` query string parameter to change the number of results returned for each request.

Use the `page` query string parameter to retrieve a particular page of results.

```
http://<Tower server name>/api/v1/model_verbose_name?page_size=100&page=2
```

The previous and next links returned with the results will set these query string parameters automatically.

The serializer is quite efficient, but you should probably not request page sizes beyond a couple of hundred.

The user interface uses smaller values to avoid the user having to do a lot of scrolling.

EIGHT

READ-ONLY FIELDS

Certain fields in the REST API are marked read-only. These usually include the URL of a resource, the ID, and occasionally some internal fields. For instance, the `created\_by` attribute of each object indicates which user created the resource, and cannot be edited.

If you post some values and notice that they are not changing, these fields may be read-only.

INTRODUCTION TO TOWER-CLI

tower-cli is a command line tool for Ansible Tower. It allows Tower commands to be easily run from the UNIX command line. It can also be used as a client library for other python apps, or as a reference for others developing API interactions with Tower's REST API.

Note: The `tower-cli` software is an open source project currently under development and, until a complete implementation occurs, only implements a subset of Tower's features.

9.1 License

While Tower is commercially licensed software, tower-cli is an open source project. Specifically, this project is licensed under the Apache 2.0 license. Pull requests, contributions, and tickets filed in GitHub are warmly welcomed.

9.2 Capabilities

`tower-cli` sends commands to the Tower API. It is capable of retrieving, creating, modifying, and deleting most objects within Tower.

A few potential uses include:

- Launching playbook runs (for instance, from Jenkins, TeamCity, Bamboo, etc)

- Checking on job statuses

- Rapidly creating objects like organizations, users, teams, and more

9.3 Installation

`tower-cli` is available as a package on PyPI.

The preferred way to install is through `pip`:

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

The main branch of this project may also be consumed directly from source.

For more information on `tower-cli`, refer to the project page at:

https://github.com/ansible/tower-cli/

9.4 Configuration

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

9.4.1 Set configuration with tower-cli config

The preferred way to set configuration is with the `tower-cli config` command.

```
$ tower-cli config key value
```

By issuing the `tower-cli config` command without arguments, you can view a full list of configuration options and where they are set.

You will generally need to set at least three configuration options–host, username, and password–which correspond to the location of your Ansible Tower instance and your credentials to authenticate to Tower.

```
$ tower-cli config host tower.example.com
$ tower-cli config username leeroyjenkins
$ tower-cli config password myPassw0rd
```

9.4.2 Write to the config files directly.

The configuration file can also be edited directly. A configuration file is a simple file with keys and values, separated by `:` or `=`:

```
host: tower.example.com
username: admin
password: p4ssw0rd
```

9.4.3 File Locations

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

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

9.4.4 Usage

`tower-cli` invocation generally follows this format:

```
$ tower-cli {resource} {action} ...
```

The **resource** is a type of object within Tower (a noun), such as `user`, `organization`, `job_template`, etc.; resource names are always singular in Tower CLI (use `tower-cli user`, never `tower-cli users`).

The **action** is the thing you want to do (a verb). Most `tower-cli` resources have the following actions–`get`, `list`, `create`, `modify`, and `delete`–and have options corresponding to fields on the object in Tower.

Examples of actions and resources include (but are not limited to):

User Actions

```
# List all users.
$ tower-cli user list

# List all non-superusers
$ tower-cli user list --is-superuser=false

# Get the user with the ID of 42.
$ tower-cli user get 42

# Get the user with the given username.
$ tower-cli user get --username=guido

# Create a new user.
$ tower-cli user create --username=guido --first-name=Guido \
                        --last-name="Van Rossum" --email=guido@python.org

# Modify an existing user.
# This would modify the first name of the user with the ID of "42" to "Guido".
$ tower-cli user modify 42 --first-name=Guido

# Modify an existing user, lookup by username.
# This would use "username" as the lookup, and modify the first name.
# Which fields are used as lookups vary by resource, but are generally
# the resource's name.
$ tower-cli user modify --username=guido --first-name=Guido

# Delete a user.
$ tower-cli user delete 42
```

Job Actions

```
# Launch a job.
$ tower-cli job launch --job-template=144

# Monitor a job.
$ tower-cli job monitor 95
```

Group Actions

```
# Get a list of groups.
$ tower-cli group --list.

# Sync a group by the groupID.
$ tower-cli group sync $groupID
```

When in doubt, check the help for more options:

```
$ tower-cli # help
$ tower-cli user --help # resource specific help
$ tower-cli user create --help # command specific help
```

9.4.5 Specify extra variables

There are a number of ways to pass extra variables to the Tower server when launching a job:

- Pass data in a file using the flag `--extra-vars="@filename.yml"`
- Include yaml data at runtime with the flag `--extra-vars="var:  value"`
- A command line editor automatically pops up when the job template is marked to prompt on launch
- If the job template has extra variables, these are not over-ridden

These methods can also be combined. For instance, if you give the flag multiple times on the command line, specifying a file in addition to manually giving extra variables, these two sources are combined and sent to the Tower server.

```
# Launch a job with extra variables from filename.yml, and also a=5
$ tower-cli job launch --job-template=1 --extra-vars="a=5 b=3" \
                                        --extra-vars="@filename.yml"

# Create a job template with that same set of extra variables
$ tower-cli job_template create --name=test_job_template --project=1 \
                             --inventory=1 --playbook=helloworld.yml \
                             --machine-credential=1 --extra-vars="a=5 b=3" \
                             --extra-vars="@filename.yml"
```

You may not combine multiple sources when modifying a job template. Whitespace can be used in strings like `--extra-vars="a='white space'"`, and list-valued parameters can be sent as JSON or YAML, but not `key=value` pairs. For instance, `--extra-vars="a:  [1, 2, 3, 4, 5]"` sends the parameter `"a"` with that list as its value.

9.4.6 SSL warnings

By default, `tower-cli` raises an error if the SSL certificate of the Tower server cannot be verified. To allow un-verified SSL connections, set the config variable,''verify_ssl'', to true. To allow it for a single command, add the `--insecure` flag.

```
# Disable insecure connection warnings permanently
$ tower-cli config verify_ssl false

# Disable insecure connection warnings for just this command
$ tower-cli job_template list --insecure
```

LAUNCHING A JOB TEMPLATE

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

Launching a Job Template also:

- Creates a Job Record

- Gives that Job Record all of the attributes on the Job Template, combined with certain data you can give in this launch endpoint ("runtime" data)

- Runs Ansible with the combined data from the JT and runtime data

Runtime data takes precedence over the Job Template data, contingent on the `ask_ _on_launch` field on the job template being set to True. For example, a runtime credential is only accepted if the Job Template has `ask_credential_on_launch` set to True.

Launching from Job Templates via the API follows the following workflow:

- GET `https://your.tower.server/api/v1/job_templates/<your job template id>/ launch/`. The response will contain data such as `job_template_data` and `defaults` which give information about the job template.

- Inspect returned data for runtime data that is needed to launch. Inspecting the OPTIONS of the launch endpoint may also help deduce what POST fields are allowed.

> **Warning:** Providing certain runtime credentials could introduce the need for a password not listed in `passwords_needed_to_start`.

  - `passwords_needed_to_start`: List of passwords needed

  - `credential_needed_to_start`: Boolean

  - `inventory_needed_to_start`: Boolean

  - `variables_needed_to_start`: List of fields that need to be passed inside of the `extra_vars` dictionary

- Inspect returned data for optionally allowed runtime data that the user should be asked for.

  - `ask_variables_on_launch`: Boolean specifying whether to prompt the user for additional variables to pass to Ansible inside of *extra_vars*

  - `ask_tags_on_launch`: Boolean specifying whether to prompt the user for `job_tags` on launch (allow allows use of `skip_tags` for convienience)

  - `ask_job_type_on_launch`: Boolean specifying whether to prompt the user for `job_type` on launch

- `ask_limit_on_launch`: Boolean specifying whether to prompt the user for `limit` on launch

- `ask_inventory_on_launch`: Boolean specifying whether to prompt the user for the related field `inventory` on launch

- `ask_credential_on_launch`: Boolean specifying whether to prompt the user for the related field `credential` on launch

- `survey_enabled`: Boolean specifying whether to prompt the user for additional `extra_vars`, following the job template's `survey_spec` Q&A format

- POST `https://your.tower.server/api/v1/job_templates/<your job template id>/launch/` with any required data gathered during the previous step(s). The variables that can be passed in the request data for this action include the following.

  - `extra_vars`: A string that represents a JSON or YAML formatted dictionary (with escaped parentheses) which includes variables given by the user, including answers to survey questions

  - `tags`: A string that represents a comma-separated list of tags in the playbook to run

  - `limit`: A string that represents a comma-separated list of hosts or groups to operate on

  - `inventory`: A integer value for the foreign key of an inventory to use in this job run

  - `credential`: A integer value for the foreign key of a credential to use in this job run

The POST will return data about the job and information about whether the runtime data was accepted. The job id is given in the `job` field to maintain compatibility with tools written before 3.0. The response will look similar to:

```
{
  "ignored_fields": {
    "credential": 2,
    "job_tags": "setup,teardown"
  }
  "id": 4,
  ...more data about the job...
  "job": 4,
}
```

In this example, values for `credential` and `job_tags` were given while the job template `ask_credential_on_launch` and `ask_tags_on_launch` were False. These were rejected because the job template author did not allow using runtime values for them.

You can see details about the job in this response. To get an updated status, you will need to do a GET request to the job page, `/jobs/4`, or follow the `url` link in the response. You can also find related links to cancel, relaunch, and so fourth.

Note: You cannot assign a new inventory at the time of launch to a scan job. Scan jobs must be tied to a fixed inventory.

Note: You cannot change the Job Type at the time of launch to or from the type of "scan". The `ask_job_type_on_launch` option only enables you to toggle "run" versus "check" at launch time.

10.1 `tower-cli` Job Template Launching

From the Tower command line, you can use `tower-cli` as a method of launching your Job Templates.

For help with `tower-cli` launch, use:

```
tower-cli job launch --help.
```

For launching from a job template, invoke tower-cli in a way similar to:

For an example of how to use the API, you can also add the `-v` flag here:

```
tower-cli job launch --job-template=4 -v
```

INDEX

- genindex

COPYRIGHT © 2016 RED HAT, INC.

Symbols