Ansible Tower API Guide

Release Ansible Tower 2.2.1

Ansible, Inc.

August 12, 2015

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

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 2.2.1.

Ansible Tower Version 2.2.1; Aug 12, 2015; http://support.ansible.com/

CHAPTER

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:

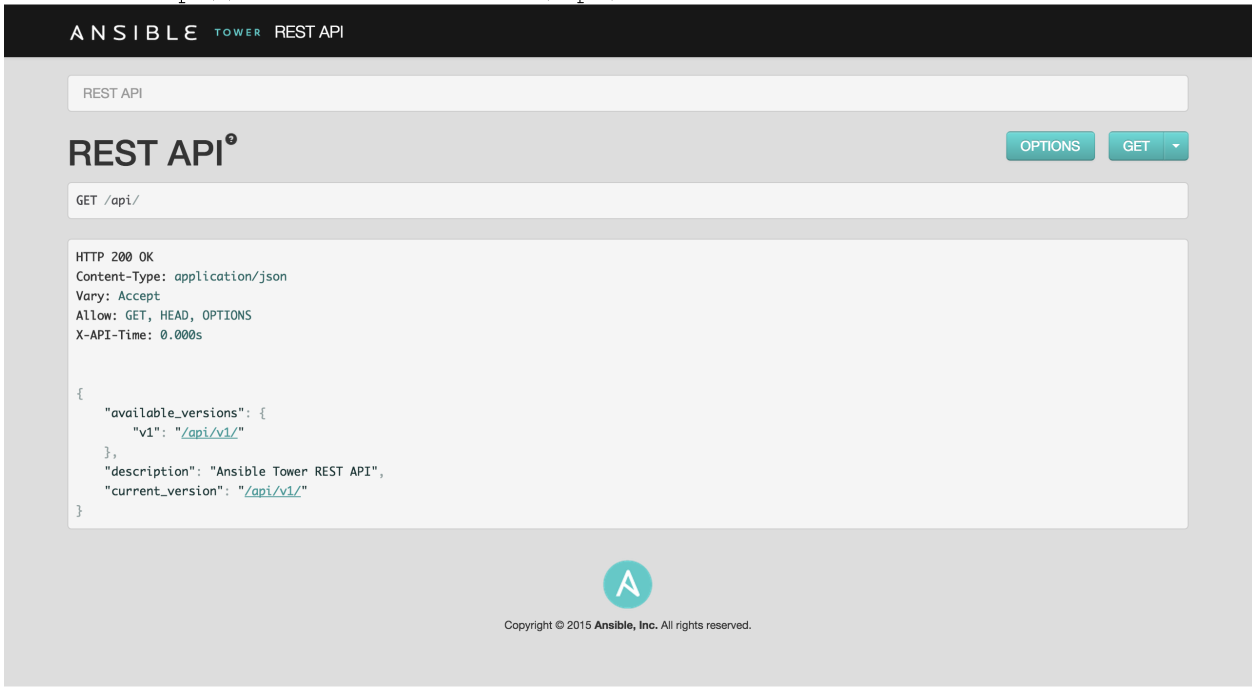
- Fiddler (http://www.telerik.com/fiddler)
- mitmproxy (https://mitmproxy.org/)
- Live HTTP headers FireFox extension (https://addons.mozilla.org/en-US/firefox/addon/live-http-headers/)
- Paros (http://sourceforge.net/projects/paros/)

CHAPTER

TWO

BROWSEABLE API

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



REST API

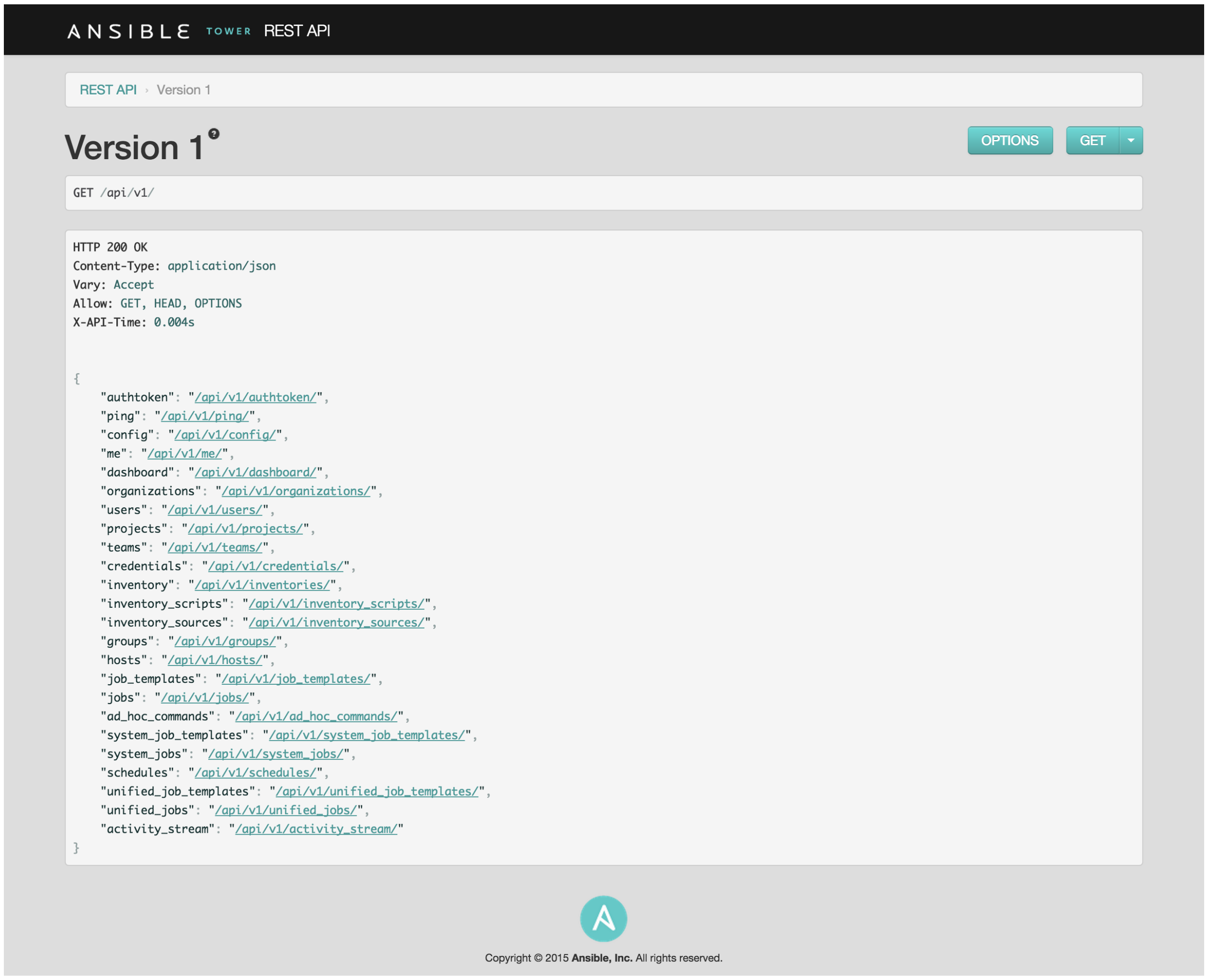
```
GET /api/
```

```
HTTP 200 OK
Content-Type: application/json
Vary: Accept
Allow: GET, HEAD, OPTIONS
X-API-Time: 0.000s


{
    "available_versions": {
        "v1": "/api/v1/"
    },
    "description": "Ansible Tower REST API",
    "current_version": "/api/v1/"
}
```

Copyright © 2015 **Ansible, Inc.** All rights reserved.

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



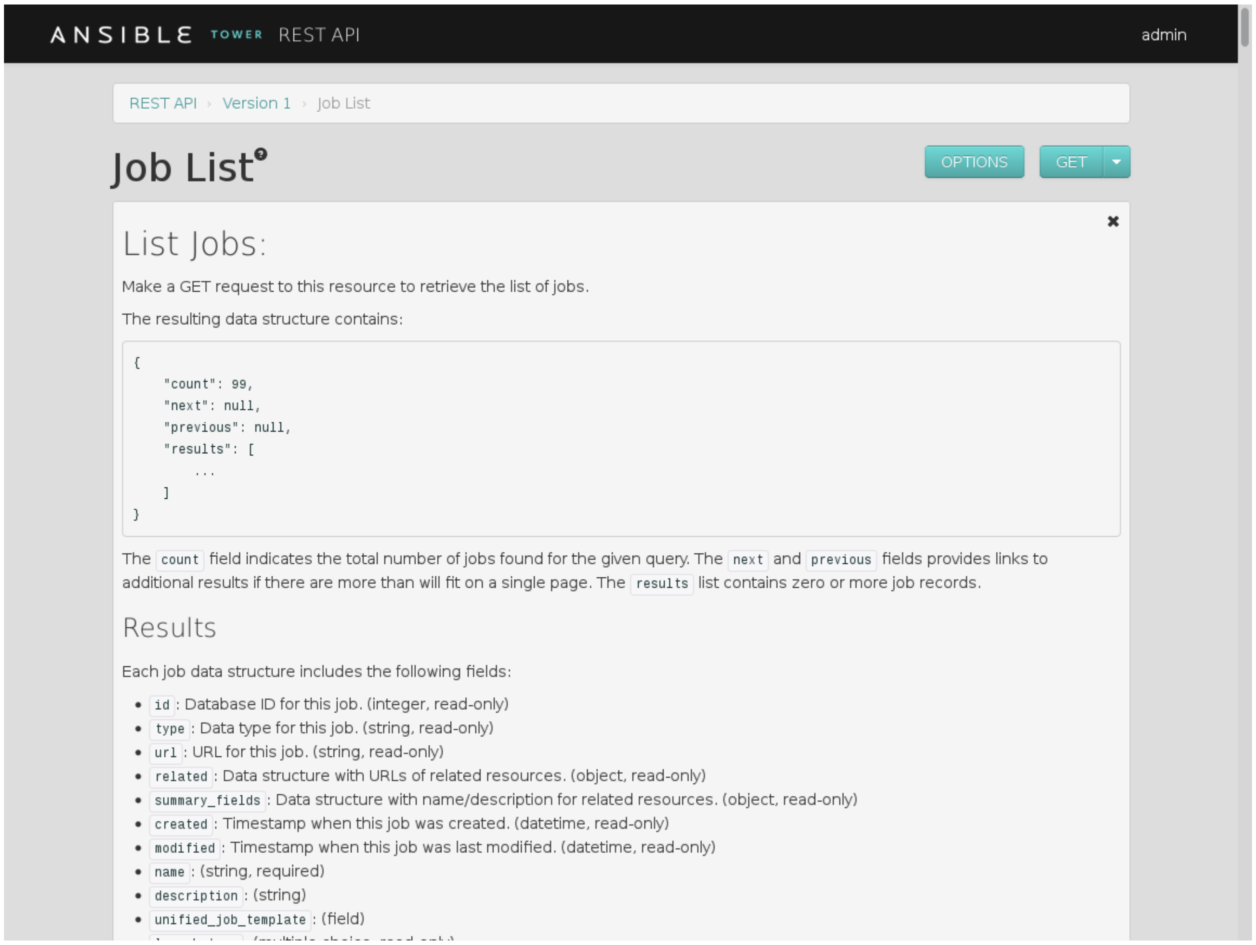
```
GET /api/v1/
```

```
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/"
}
```

Copyright © 2015 **Ansible, Inc.** All rights reserved.

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.



Job List

List Jobs:

Make a GET request to this resource to retrieve the list of jobs.

The resulting data structure contains:

```
{
    "count": 99,
    "next": null,
    "previous": null,
    "results": [
        ...
    ]
}
```

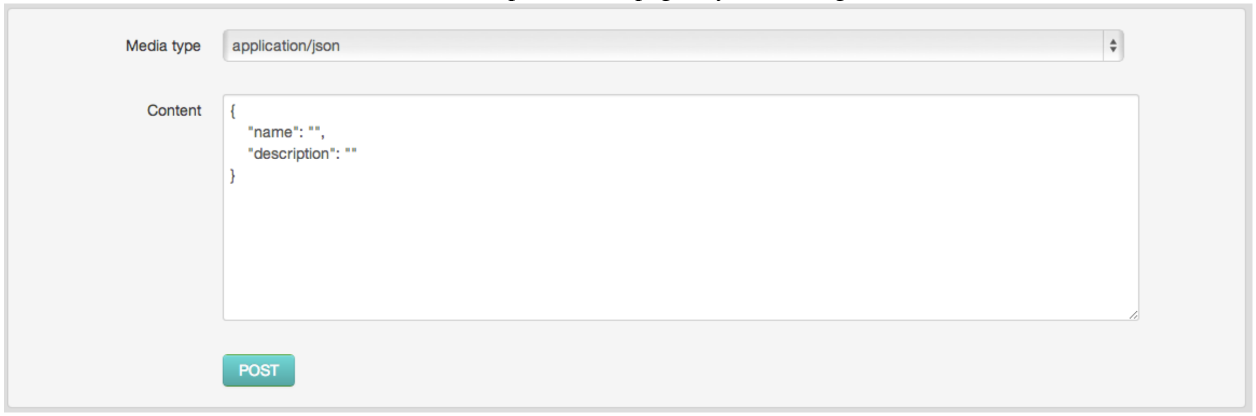
The `count` field indicates the total number of jobs found for the given query. The `next` and `previous` fields provides links to additional results if there are more than will fit on a single page. The `results` list contains zero or more job records.

Results

Each job data structure includes the following fields:

- `id` : Database ID for this job. (integer, read-only)
- `type` : Data type for this job. (string, read-only)
- `url` : URL for this job. (string, read-only)
- `related` : Data structure with URLs of related resources. (object, read-only)
- `summary_fields` : Data structure with name/description for related resources. (object, read-only)
- `created` : Timestamp when this job was created. (datetime, read-only)
- `modified` : Timestamp when this job was last modified. (datetime, read-only)
- `name` : (string, required)
- `description` : (string)
- `unified_job_template` : (field)

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



Media type: application/json

Content:

```
{
    "name": "",
    "description": ""
}
```

CHAPTER

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 2.2.1 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.

CHAPTER FOUR

SORTING

Assume the following URL, `http://<Tower server name>/api/v1/groups/`

In order to sort the groups by name, access the following URL variation:

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

You can order by any field in the object.

CHAPTER

FIVE

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 do 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 more than one field at once:

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

This will find 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, see:

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

You may also wish to watch the API as the UI is being used to see how it is filtering on various criteria.

CHAPTER

SIX

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 like:

```
{'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.

To request more items per page, pass the page size query string:

```
http://<Tower server name>/api/v1/some_resource?page_size=50
```

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.

CHAPTER

SEVEN

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.

CHAPTER

EIGHT

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.

8.1 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

8.2 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/

CHAPTER NINE

INDEX

- genindex