
Automation Controller Administration Guide

Release Automation Controller 4.0.1

Red Hat, Inc.

Dec 09, 2022

CONTENTS

1	Red Hat Ansible Automation Platform controller Licensing, Updates, and Support	2
1.1	Support	2
1.2	Trial / Evaluation	2
1.3	Subscription Types	2
1.4	Node Counting in Licenses	3
1.5	Attaching Subscriptions	3
1.6	Ansible Automation Platform Component Licenses	4
2	Starting, Stopping, and Restarting the Controller	5
3	Custom Inventory Scripts	6
4	Inventory File Importing	7
4.1	Custom Dynamic Inventory Scripts	7
4.2	SCM Inventory Source Fields	9
5	Multi-Credential Assignment	10
5.1	Background	10
5.2	Important Changes	10
5.3	Launch Time Considerations	11
5.4	Multi-Vault Credentials	11
6	Management Jobs	14
6.1	Removing Old Activity Stream Data	15
6.2	Cleanup Expired OAuth2 Tokens	18
6.3	Cleanup Expired Sessions	18
6.4	Removing Old Job History	18
7	Clustering	20
7.1	Setup Considerations	20
7.2	Install and Configure	21
7.3	Status and Monitoring via Browser API	22
7.4	Instance Services and Failure Behavior	22
7.5	Job Runtime Behavior	23
7.6	Deprovision Instances	24
8	Container and Instance Groups	25
8.1	Instance Groups	25
8.2	Container Groups	31
9	Controller Logfiles	37

10	Logging and Aggregation	39
10.1	Loggers	40
10.2	Set Up Logging	44
10.3	Troubleshoot Logging	46
11	Metrics	47
11.1	Set up Prometheus	47
12	Secret handling and connection security	49
12.1	Secret Handling	49
12.2	Connection Security	50
13	Security Best Practices	52
13.1	General best practices	52
13.2	Understand the architecture of Ansible and the controller	52
13.3	Granting access	53
13.4	Available resources	54
14	The <i>awx-manage</i> Utility	56
14.1	Inventory Import	56
14.2	Cleanup of old data	57
14.3	Cluster management	57
14.4	Token and session management	57
14.5	Analytics gathering	59
15	Controller Configuration	60
15.1	Authentication	60
15.2	Jobs	61
15.3	System	62
15.4	User Interface	63
15.5	License	66
16	Isolation functionality and variables	70
17	Token-Based Authentication	71
17.1	Managing OAuth 2 Applications and Tokens	71
17.2	Using OAuth 2 Token System for Personal Access Tokens (PAT)	74
17.3	Application Functions	76
17.4	Application Token Functions	78
18	Setting up Social Authentication	81
18.1	GitHub settings	81
18.2	Google OAuth2 settings	87
18.3	Organization and Team Mapping	89
19	Setting up Enterprise Authentication	91
19.1	Azure AD settings	91
19.2	LDAP Authentication	92
19.3	RADIUS settings	92
19.4	SAML settings	93
19.5	TACACS+ settings	101
20	Setting up LDAP Authentication	102
20.1	Referrals	107
20.2	Enabling Logging for LDAP	108
20.3	LDAP Organization and Team Mapping	108

21	Changing the Default Timeout for Authentication	110
22	User Authentication with Kerberos	111
22.1	AD and Kerberos Credentials	112
22.2	Working with Kerberos Tickets	113
23	Working with Session Limits	114
24	Backing Up and Restoring	115
24.1	Backup/Restore Playbooks	116
24.2	Backup and Restoration Considerations	116
24.3	Backup and Restore for Clustered Environments	117
25	Using Custom Logos in automation controller	118
26	Usability Analytics and Data Collection	120
26.1	Automation Analytics	120
26.2	Details of data collection	125
27	Troubleshooting the controller	136
27.1	Error logs	136
27.2	sosreport	136
27.3	Problems connecting to your host	136
27.4	Unable to login to the controller via HTTP	137
27.5	WebSockets port for live events not working	137
27.6	Problems running a playbook	137
27.7	Problems when running a job	137
27.8	Playbooks aren't showing up in the "Job Template" drop-down	138
27.9	Playbook stays in pending	138
27.10	Cancel a controller job	138
27.11	Reusing an external database causes installations to fail	138
27.12	Private EC2 VPC Instances in the controller Inventory	139
27.13	Troubleshooting "Error: provided hosts list is empty"	140
28	Controller Tips and Tricks	141
28.1	Using the Controller CLI Tool	141
28.2	Changing the Controller Admin Password	142
28.3	Creating a controller Admin from the commandline	142
28.4	Setting up a jump host to use with the controller	142
28.5	View Ansible outputs for JSON commands when using the controller	143
28.6	Locate and configure the Ansible configuration file	143
28.7	View a listing of all ansible_ variables	143
28.8	The ALLOW_JINJA_IN_EXTRA_VARS variable	143
28.9	Using execution environments	144
28.10	Configuring the controllerhost hostname for notifications	144
28.11	Launching Jobs with curl	144
28.12	Dynamic Inventory and private IP addresses	145
28.13	Filtering instances returned by the dynamic inventory sources in the controller	145
28.14	Using an unreleased module from Ansible source with the controller	146
28.15	Using callback plugins with the controller	146
28.16	Connecting to Windows with winrm	147
28.17	Importing existing inventory files and host/group vars into the controller	147
29	Postface	149

30 Index	152
31 Copyright © Red Hat, Inc.	153
Index	154

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

The *Automation Controller Administration Guide* documents the administration of automation controller through custom scripts, management jobs, and more. Written for DevOps engineers and administrators, the *Automation Controller Administration Guide* assumes a basic understanding of the systems requiring management with the controller's easy-to-use graphical interface. This document has been updated to include information for the latest release of Automation Controller v4.0.1.

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!

Automation Controller Version 4.0.1; April 21, 2022; <https://access.redhat.com/>

RED HAT ANSIBLE AUTOMATION PLATFORM CONTROLLER LICENSING, UPDATES, AND SUPPORT

Red Hat Ansible Automation Platform controller (“**Automation Controller**”) 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>

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 automation controller 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 automation controller 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 automation controller, 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 Red Hat Ansible Automation Platform controller 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

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 **[default]** 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
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 Ansible Automation Platform Component Licenses

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

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

STARTING, STOPPING, AND RESTARTING THE CONTROLLER

Automation controller ships with an *admin utility script*, `automation-controller-service`, that can start, stop, and restart all the controller services running on the current single controller 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/automation-controller-service` and can be invoked as follows:

```
root@localhost:~$ automation-controller-service restart
```

Note: In clustered installs, `automation-controller-service restart` does not include PostgreSQL as part of the services that are restarted because it exists external to the controller, and because PostgreSQL does not always require a restart. Use `systemctl restart automation-controller` 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 the controller in a container, do not use the `automation-controller-service` script. Restart the pod using the container environment instead.

CUSTOM INVENTORY SCRIPTS

Inventory scripts have been discontinued. For more information, see [Export old inventory scripts](#) in the *Automation Controller User Guide*.

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 *Automation Controller User Guide* for more detail.

INVENTORY FILE IMPORTING

The controller 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.

Any imported hosts will have a description of “imported” by default. This can be overridden by setting the `_awx_description` variable on a given host. For example, if importing from a sourced `.ini` file, you could add the following host variables:

```
[main]
127.0.0.1 _awx_description="my host 1"
127.0.0.2 _awx_description="my host 2"
```

Similarly, group descriptions also default to “imported”, but can be overridden by the `_awx_description` as well.

In order to use old inventory scripts in source control, see [Export old inventory scripts](#) in the *Automation Controller User Guide* for detail.

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 the controller, 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 > Demo Inventory > Sources

Create new source

The screenshot shows the 'Create new source' configuration page. At the bottom, under 'Update options', there are four checkboxes: 'Overwrite', 'Overwrite variables', 'Update on launch', and 'Update on project update'. The 'Update on project update' checkbox is checked and has a red arrow pointing to it.

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 controller User Interface from the Create Inventory Source page. Refer to the [Inventories](#) section of the *Automation Controller 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.

Jobs > Project from Git 🔍

Output

← Back to Jobs Details **Output**

Project from Git Plays 2 Tasks 7 Hosts 1 Elapsed 00:00:05

Stdout

```

0  WARN[0000] error mounting subscriptions, skipping entry in /usr/share/containers/mounts.conf: getting host subscription data failed: failed to read subscriptions from "/usr/share/rhel/s
1  ecrets": open /usr/share/rhel/secrets/rhsm/syspurpose/syspurpose.json: permission denied
2  PLAY [Update source tree if necessary] ***** 10:55:31
3
4  TASK [update project using git] ***** 10:55:31
5  ok: [localhost]
6
7  TASK [Set the git repository version] ***** 10:55:32
8  ok: [localhost]
9
10 TASK [Repository Version] ***** 10:55:32
11 ok: [localhost] => {
12   "msg": "Repository Version d357156bbad59cdea641d48dab99140e033fd089"
13 }
14
15 PLAY [Install content with ansible-galaxy command if necessary] ***** 10:55:32

```

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

4.2.1 Supported File Syntax

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

MULTI-CREDENTIAL ASSIGNMENT

automation controller provides support for assigning zero or more credentials to a job template.

5.1 Background

Prior to automation controller 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 automation controller 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 the controller 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 controller User Interface in the *Create Credential* page:

CREENTIALS / CREATE CREDENTIAL ⌵

NEW CREDENTIAL ✕

DETAILS | PERMISSIONS

* NAME ? DESCRIPTION ? ORGANIZATION ?

* CREDENTIAL TYPE ?

TYPE DETAILS

* VAULT PASSWORD Prompt on launch VAULT IDENTIFIER ?

SHOW

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:

The screenshot shows the 'NEW JOB TEMPLATE' configuration page. The 'CREDENTIALS' field is selected with a dropdown menu showing 'Multi-Vault Credential | first'. A red arrow points to this selection. Other fields include NAME, INVENTORY, PROJECT, VERBOSITY, LABELS, TIMEOUT, FORKS, JOB TAGS, INSTANCE GROUPS, SHOW CHANGES, JOB TYPE, PLAYBOOK, LIMIT, SKIP TAGS, JOB SLICING, and various options like ENABLE PRIVILEGE ESCALATION, etc.

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:

The screenshot shows the 'NEW CREDENTIAL' configuration page. The 'VAULT IDENTIFIER' field is circled in red with a speech bubble that says 'Leave blank'. Other fields include NAME, DESCRIPTION, ORGANIZATION, CREDENTIAL TYPE, VAULT PASSWORD, and SHOW.

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.

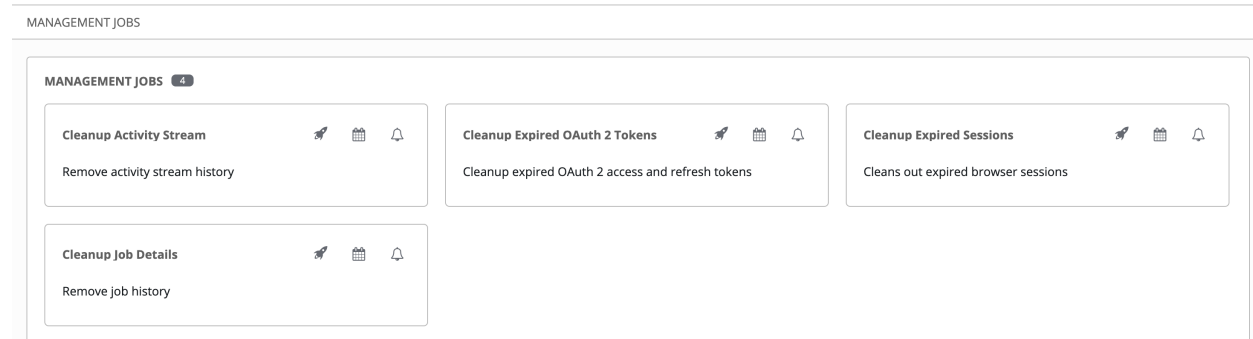
```
POST /api/v2/job_templates/N/launch/
{
  'credential_passwords': {
    'vault_password.X': 'first-vault-password'
    'vault_password.Y': 'second-vault-password'
  }
}
```

5.4.2 Linked credentials

Instead of uploading sensitive credential information into the controller, you can link credential fields to external systems and using them to run your playbooks. Refer to the [Secret Management System](#) section of the *Automation Controller User Guide*.

MANAGEMENT JOBS


Management Jobs assist in the cleaning of old data from the controller, 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 controller database. Click **Management Jobs** from the left navigation bar.

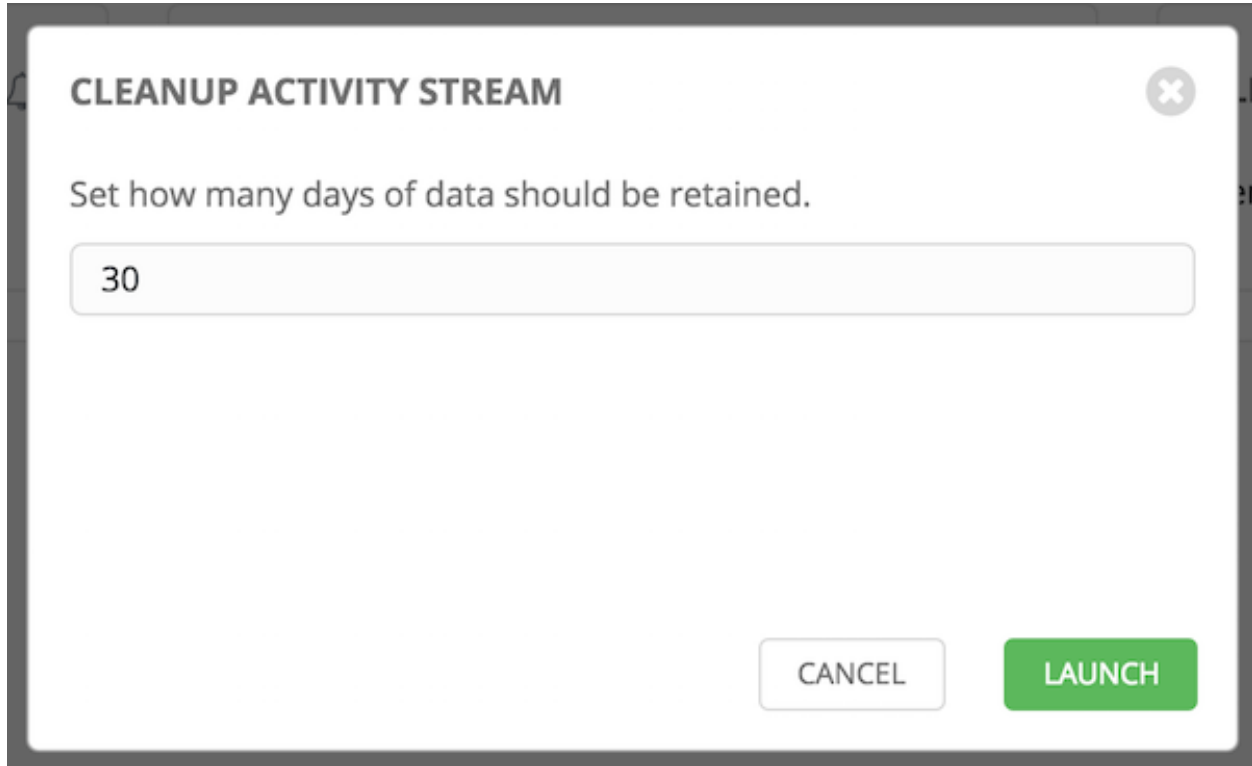


Several job types are available for you to schedule and launch:

- **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

CANCEL LAUNCH

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				
SEARCH <input type="text"/> <input type="button" value="Q"/>				
KEY <input type="text"/> <input type="button" value="+"/>				
NAME ^	FIRST RUN ↕	NEXT RUN ↕	FINAL RUN ↕	ACTIONS
<input checked="" type="checkbox"/> Cleanup Activity Schedule	12/17/2019 1:44:17 PM	2/4/2020 1:44:17 PM		<input type="button" value="edit"/> <input type="button" value="delete"/>
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 **Add** button to create a new schedule for this management job.

MANAGEMENT JOBS / SCHEDULES / EDIT SCHEDULED JOB

Cleanup Activity Schedule ✕

* NAME

* LOCAL TIME ZONE

* START DATE

* REPEAT FREQUENCY

* START TIME (HH24:MM:SS)
 : :

* DAYS OF DATA TO KEEP

FREQUENCY DETAILS

* EVERY
 WEEKS

* ON DAYS
 SUN MON TUE WED THU FRI SAT

* END

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 ✕

<p>* NAME</p> <input type="text" value="Clean up Activity Stream - Slack"/>	<p>DESCRIPTION</p> <input type="text" value="Slack notification for activity stream management job"/>	<p>* ORGANIZATION</p> <input type="text" value="Honey Dog, Inc."/>
<p>* TYPE</p> <input type="text" value="Slack"/>		
<p>TYPE DETAILS</p>		
<p>* DESTINATION CHANNELS ⓘ</p> <input type="text" value="#engineering
#rel-eng"/>	<p>* TOKEN</p> <p>SHOW <input type="text" value="....."/></p>	<p>NOTIFICATION COLOR ⓘ</p> <input type="text" value="Green"/>

Refer to [Notifications](#) in the *Automation Controller 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 *Automation Controller User Guide* for more detail.


6.3 Cleanup Expired Sessions

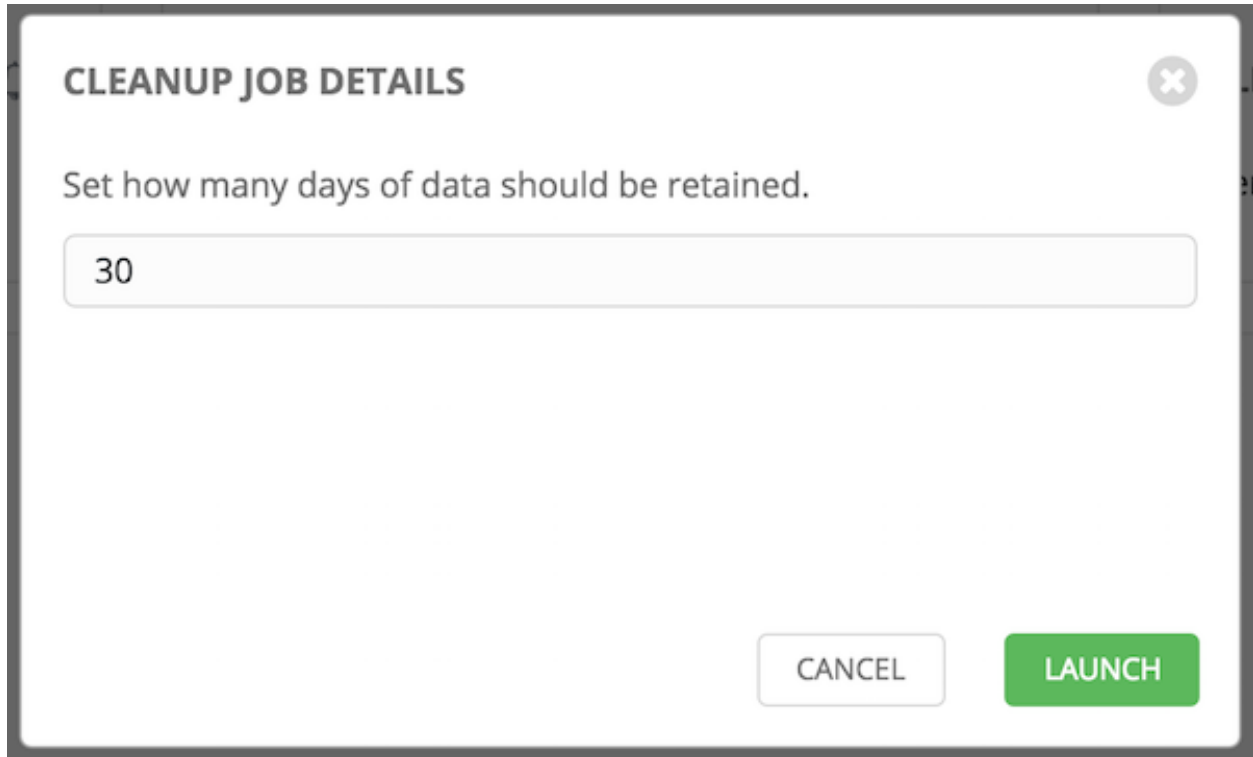
To remove expired sessions, click on the launch () button beside **Cleanup Expired Sessions**.

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 *Automation Controller 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.

CANCEL LAUNCH

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

Note: The initial job run for a controller 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 *Automation Controller User Guide* for more detail.

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 the controller 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 controller 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*.

Ansible Automation Platform supports container-based clusters using Kubernetes, meaning new controller instances can be installed on this platform without any variation or diversion in functionality. You can create instance groups to point to a Kubernetes container. For more detail, see the *Container and Instance Groups* 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 automation controller 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 Automation Platform Upgrade and Migration Guide*.

Important considerations to note in the new clustering environment:

- PostgreSQL is still a standalone instance and is not clustered. The controller 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 controller 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 controller 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 `default` group in the inventory *AND* it needs to be the first host listed in the `default` group.
- Manual projects must be manually synced to all instances by the customer, and updated on all instances at once.
- The `inventory` file for platform 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 contains a single inventory group, `automationcontroller`.

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.

```
[automationcontroller]
hostA
hostB
hostC

[instance_group_east]
hostB
hostC

[instance_group_west]
hostC
hostD
```

Note: If no groups are selected for a resource then the `automationcontroller` group is used, but if any other group is selected, then the `automationcontroller` 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:

```
[automationcontroller]
hostA
hostB
hostC

[database]
hostDB
```

When a playbook runs on an individual controller instance in a cluster, the output of that playbook is broadcast to all of the other nodes as part of the controller'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:

```
[automationcontroller]
hostA routable_hostname=10.1.0.2
```

(continues on next page)

(continued from previous page)

```
hostB routable_hostname=10.1.0.3
hostC routable_hostname=10.1.0.4
```

Note: Prior versions of automation controller used the variable name `rabbitmq_host`. If you are upgrading from a previous version of the platform, 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 the Controller and Automation Hub

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

- 80, 443 (normal controller 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 controller instances)

7.3 Status and Monitoring via Browser API

The controller 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 controller instance is made up of several different services working collaboratively:

- HTTP Services - This includes the controller 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.

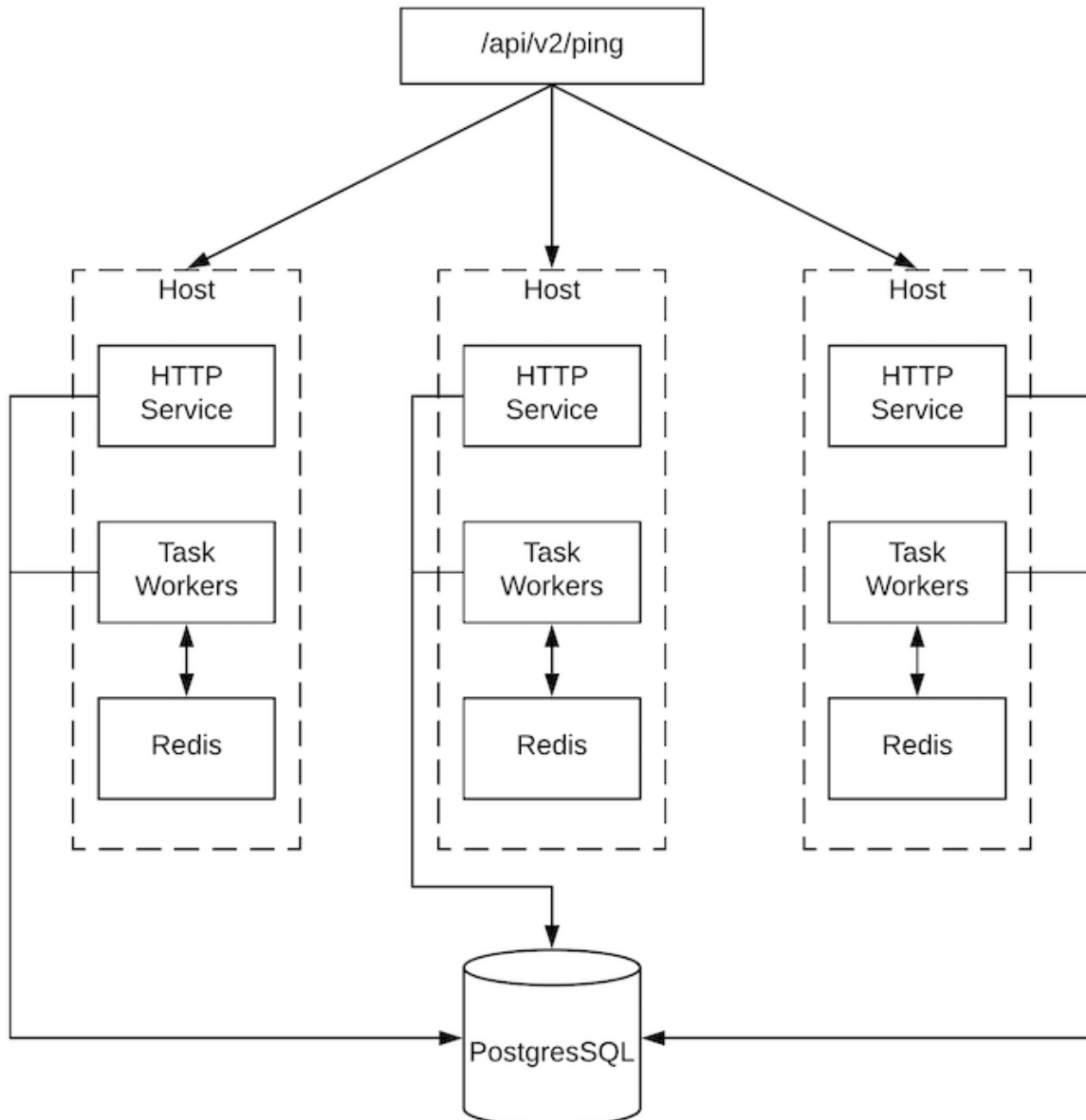
The controller is configured in such a way that if any of these services or their components fail, then all services are restarted. If these fail sufficiently often in a short span of time, then the entire instance will be placed offline in an automated fashion in order to allow remediation without causing 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 controller 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 controller 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 controller 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 controller instance and then run the deprovisioning tool from any other instance:

1. Shut down the instance or stop the service with the command, `automation-controller-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 controller cluster.

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

Similarly, deprovisioning instance groups in the controller does not automatically deprovision or remove instance groups. For more information, refer to the [Deprovision Instance Groups](#) section.

CONTAINER AND INSTANCE GROUPS

The controller allows you to execute jobs via ansible playbook runs directly on a member of the cluster or in a namespace of an Openshift cluster with the necessary service account provisioned called a Container Group. 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.

For execution environments, see [Execution Environments](#) in the *Automation Controller User Guide*.

8.1 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 required to be in the `automationcontroller` or `execution_nodes` group alongside other `instance_group_` groups. In a clustered setup, at least one instance **must** be present in the `automationcontroller` group, which will appear as `controlplane` in the API instance groups. See [automationcontroller group policies](#) for example scenarios.
- A default API instance group is automatically created with all nodes capable of running jobs. Technically, it is like any other instance group but if a specific instance group is not associated with a specific resource, then job execution will always fall back to the default instance group. The default instance group always exists (it cannot be deleted nor renamed).
- Do not create a group named `instance_group_default`.
- Do not name any instance the same as a group name.

8.1.1 automationcontroller group policies

Use the following criteria when defining nodes:

- nodes in the `automationcontroller` group can define `node_type` hostvar to be `hybrid` (default) or `control`
- nodes in the `execution_nodes` group can define `node_type` hostvar to be `execution` (default) or `hop`

You can define custom groups in the inventory file by naming groups with `instance_group_*` where `*` becomes the name of the group in the API. Or, you can create custom instance groups in the API after the install has finished.

The current behavior expects a member of an `instance_group_*` be part of `automationcontroller` or `execution_nodes` group. Consider this example scenario:

```
[automationcontroller]
126-addr.tatu.home ansible_host=192.168.111.126 node_type=control

[automationcontroller:vars]
peers=execution_nodes

[execution_nodes]

[instance_group_test]
110-addr.tatu.home ansible_host=192.168.111.110 receptor_listener_port=8928
```

As a result of running the installer, you will get the error below:

```
TASK [ansible.automation_platform_installer.check_config_static : Validate mesh_
↳topology] ***
fatal: [126-addr.tatu.home -> localhost]: FAILED! => {"msg": "The host '110-addr.tatu.
↳home' is not present in either [automationcontroller] or [execution_nodes]"}
```

To fix this, you could move the box `110-addr.tatu.home` to an `execution_node` group.

```
[automationcontroller]
126-addr.tatu.home ansible_host=192.168.111.126 node_type=control

[automationcontroller:vars]
peers=execution_nodes

[execution_nodes]
110-addr.tatu.home ansible_host=192.168.111.110 receptor_listener_port=8928

[instance_group_test]
110-addr.tatu.home
```

This results in:

```
TASK [ansible.automation_platform_installer.check_config_static : Validate mesh_
↳topology] ***
ok: [126-addr.tatu.home -> localhost] => {"changed": false, "mesh": {"110-addr.tatu.
↳home": {"node_type": "execution", "peers": [], "receptor_control_filename":
↳"receptor.sock", "receptor_control_service_name": "control", "receptor_listener":
↳true, "receptor_listener_port": 8928, "receptor_listener_protocol": "tcp",
↳"receptor_log_level": "info"}, "126-addr.tatu.home": {"node_type": "control", "peers
↳": ["110-addr.tatu.home"], "receptor_control_filename": "receptor.sock", "receptor_
↳control_service_name": "control", "receptor_listener": false, "receptor_listener_
↳port": 27199, "receptor_listener_protocol": "tcp", "receptor_log_level": "info"}}
```

Upon upgrading from controller 4.0 or earlier, the legacy `instance_group_member` will most likely have the `awx` code installed, which would cause that node to be placed in the `automationcontroller` group.

8.1.2 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.

8.1.3 Instance group policies

You can configure controller 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 controller 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 automation controller User Interface provides a summary of the capacity levels for each instance group according to instance group policies:

Instance Groups 🔍

Name	Type	Running Jobs	Total Jobs	Instances	Capacity	Actions
<input type="checkbox"/> Can't contain myself	Container group	0	0	0		
<input type="checkbox"/> controlplane	Instance group	1	15	1	Used capacity 2%	
<input type="checkbox"/> default	Instance group	0	0	2	Unavailable	
<input type="checkbox"/> test-instance-group	Instance group	0	0	2	Unavailable	

1 - 4 of 4 items << < 1 of 1 page > >>

8.1.4 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.

8.1.5 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
}
```

8.1.6 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 controller instances are brought online, it effectively expands the work capacity of the 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.

8.1.7 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 `default` 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 `default` group with its inventory, you allow the `default` group to be used as a fallback in case `group_a` gets out of capacity.

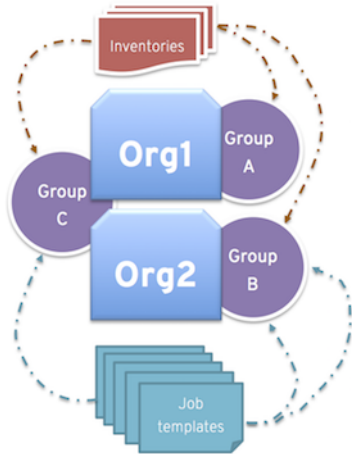
In addition, it is possible to not associate an instance group with one resource but designate another resource as the fallback. For example, not associating an instance group with a job template and have it fall back to the inventory and/or the organization's instance group.

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 controller cluster.

8.1.8 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 controller instance and then run the deprovisioning tool from any other instance:

1. Shut down the instance or stop the service with the command, `automation-controller-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 controller cluster registry.

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

Similarly, deprovisioning instance groups in the controller 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 automation controller User Interface. For more information on managing instance groups in the UI, refer to [Instance Groups](#) in the *Automation Controller User Guide*.

Note: If you have isolated instance groups created in older versions of the controller (3.8.x and earlier) and want to migrate them to execution nodes to make them compatible for use with the automation mesh architecture, see `migrate_iso_to_exe` in the *Ansible Automation Platform Upgrade and Migration Guide*.

8.2 Container Groups

Ansible Automation Platform supports [Container Groups](#), which allow you to execute jobs in the controller regardless of whether the controller 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 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 container groups be “always-on”, which is configured through the creation of an instance.

Note: Container Groups upgraded from versions prior to automation controller 4.0 will revert back to default and completely remove the old pod definition, clearing out all custom pod definitions in the migration.

Container groups are different from execution environments in that execution environments are container images and do not use a virtual environment. See [Execution Environments](#) in the *Automation Controller User Guide* for further detail.

8.2.1 Create a container group

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

- A namespace you can launch into (every cluster has a “default” namespace, but you may want to use a specific namespace)
- A service account that has the roles that allow it to launch and manage Pods in this namespace
- If you will be using execution environments in a private registry, and have a Container Registry credential associated to them in the automation controller, the service account also needs the roles to get, create, and delete secrets in the namespace. If you do not want to give these roles to the service account, you can pre-create the `ImagePullSecrets` and specify them on the pod spec for the `ContainerGroup`. In this case, the execution environment should NOT have a Container Registry credential associated, or the controller will attempt to create the secret for you in the namespace.
- A token associated with that service account (OpenShift or Kubernetes Bearer Token)
- A CA certificate associated with the cluster

This section describes creating a Service Account in an OpenShift cluster (or K8s) in order to be used to run jobs in a container group via automation controller. After the Service Account is created, its credentials are provided to the controller in the form of an OpenShift or Kubernetes API bearer token credential. Below describes how to create a service account and collect the needed information for configuring automation controller.

To configure the controller:

1. To create a service account, you may download and use this sample service account, `containergroup_sa` and modify it as needed to obtain the above credentials.
2. Apply the configuration from `containergroup-sa.yml`:

```
oc apply -f containergroup-sa.yml
```

3. Get the secret name associated with the service account:

```
export SA_SECRET=$(oc get sa containergroup-service-account -o json | jq '.
↪secrets[0].name' | tr -d '"')
```

4. Get the token from the secret:

```
oc get secret $(echo ${SA_SECRET}) -o json | jq '.data.token' | xargs | base64 --
↳ decode > containergroup-sa.token
```

5. Get the CA cert:

```
oc get secret $SA_SECRET -o json | jq '.data["ca.crt"]' | xargs | base64 --decode_
↳ > containergroup-ca.crt
```

6. Use the contents of `containergroup-sa.token` and `containergroup-ca.crt` to provide the information for the [OpenShift or Kubernetes API Bearer Token](#) required for the container group.

To create a container group:

1. Use the controller 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 *Automation Controller User Guide* for detail.
2. Create a new container group by navigating to the Instance Groups configuration window by clicking **Instance Groups** from the left navigation bar.
3. Click the **Add** button and select **Create Container Group**.

Instance Groups 🔍

Create new container group

Name *

Credential 🔍

Options Customize pod specification

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

8.2.2 Customize the Pod spec

Ansible Automation Platform 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. `ImagePullSecrets`) that can be “serialized” as valid Pod JSON or YAML. A full list of options can be found in the [OpenShift 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.

Name *

Credential 🔍

Options Customize pod specification

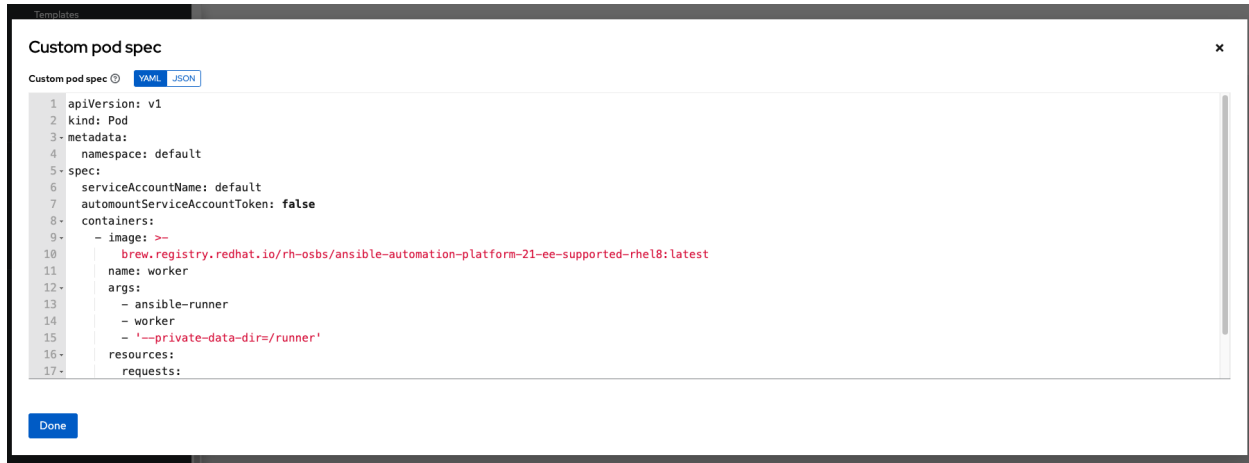
Custom pod spec 🔍 YAML JSON

```

1 apiVersion: v1
2 kind: Pod
3 - metadata:
4   namespace: default
5 - spec:
6   serviceAccountName: default

```

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




```

1 apiVersion: v1
2 kind: Pod
3 metadata:
4   namespace: default
5 spec:
6   serviceAccountName: default
7   automountServiceAccountToken: false
8   containers:
9     - image: >-
10       brew.registry.redhat.io/rh-osbs/ansible-automation-platform-21-ee-supported-rhel8:latest
11     name: worker
12     args:
13       - ansible-runner
14       - worker
15       - '--private-data-dir=/runner'
16     resources:
17       requests:

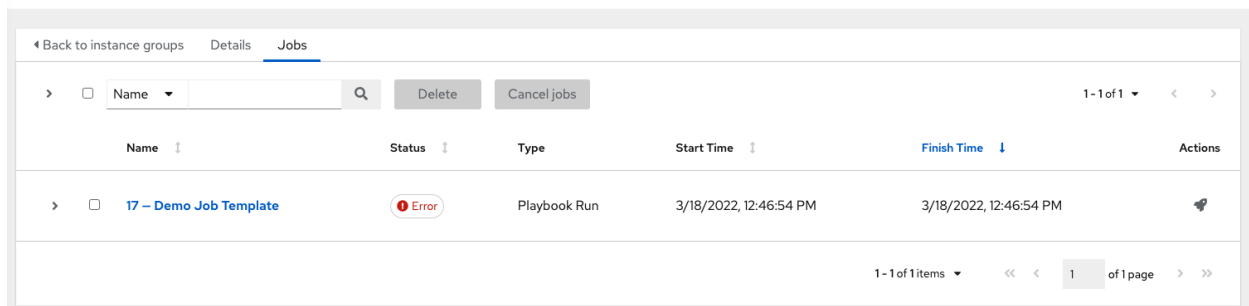
```

Note: The image used at job launch time is determined by which execution environment is associated with the job. If a Container Registry credential is associated with the execution environment, then the controller will attempt to make a `ImagePullSecret` to pull the image. If you prefer not to give the service account permission to manage secrets, you must pre-create the `ImagePullSecret` and specify it on the pod spec, and omit any credential from the execution environment used.

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.

Instance Groups > test-container-group
Jobs



Name	Status	Type	Start Time	Finish Time	Actions
17 - Demo Job Template	Error	Playbook Run	3/18/2022, 12:46:54 PM	3/18/2022, 12:46:54 PM	

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 the platform 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.

8.2.3 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 *Automation Controller User Guide* for detail.

Inventories 🔍

Create new inventory

Name * Description Organization *

Instance Groups

Variables

1 ---

2. Create “localhost” host in inventory with variables:

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

Name * Description

Variables

1 ---

2 {'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.

Inventories > Container Group Test Inventory 🔍

Hosts

◀ Back to Inventories Details Access Groups Hosts Sources Jobs

Name 1-1 of 1

Name ↑	Actions
<input checked="" type="checkbox"/> localhost	<input type="button" value="On"/> <input type="button" value="Edit"/>

1-1 of 1 items 1 of 1 page

↓

Run command

1 Details
2 Execution Environment
3 Credential
4 Preview

Module * ⓘ
ping

Arguments ⓘ
[Empty text box]

Verbosity * ⓘ
0 (Normal)

Limit ⓘ
localhost

Forks ⓘ
0

Show changes ⓘ Enable privilege escalation ⓘ
 Off

Extra variables ⓘ **YAML** JSON

Next Back Cancel

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

Jobs > ping
Output

ping ✔ Successful Elapsed 00:00:00

Stdout

```

0 Localhost | SUCCESS => {
1   "ansible_facts": {
2     "discovered_interpreter_python": "/usr/bin/python"
3   },
4   "changed": false,
5   "ping": "pong"

```

If you have an OpenShift 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.

8.2.4 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 container group and the execution environment that spun up.

Jobs > 1028 - JobTemplate - PatienceKitchen

Details

← Back to Jobs Details Output

Job ID	1028	Status	Successful	Started	3/22/2022, 10:12:11 AM
Finished	3/22/2022, 10:12:54 AM	Job Template	JobTemplate - PatienceKitchen	Job Type	Playbook Run
Launched By	admin	Inventory	Inventory - LegAddition	Project	Project - KindWin
Project Status	Successful	Revision	98b8dc2d4d6671ddceab73a5d3958e94fdbba419	Playbook	ping.yml
Verbosity	0 (Normal)	Execution Environment	AWX EE (latest)	Container Group	ContainerGroup - rsftrt4xqa
Job Slice	0/1	Created	3/22/2022, 10:12:10 AM by admin	Last Modified	3/22/2022, 10:12:10 AM

Variables [YAML](#) [JSON](#)

8.2.5 Kubernetes API failure conditions

When running a container group and the Kubernetes API responds that the resource quota has been exceeded, the controller 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:

Jobs > Demo Job Template

Output

← Back to Jobs Details Output

Demo Job Template Elapsed 00:00:04

Stdout

```
0 Error creating pod: pods is forbidden: User "system:serviceaccount:aap:example" cannot create resource "pods" in API group "" in the namespace "aap"
```

8.2.6 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 OpenShift documentation for [Quotas and Limit Ranges](#).
- To set limits directly on the pod definition launched by the controller, see [Customize the Pod spec](#) and refer to the OpenShift documentation to set the options to `compute resources`.

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 the controller, that setting will be passed along to the container.

CONTROLLER LOGFILES

The controller 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 controller dispatcher worker service.
- **job_lifecycle.log:** captures details of the job run, whether it is blocked, and what condition is blocking it.
- **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 the controller is running in the background, such as adding 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 the controller system tracking migration or upgrade.
- **wsbroadcast.log:** captures the logs of websocket connections in the controller 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 supervisor.
- **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 controller 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 the controller.
- **failure-event-handler.stderr.log:** captures the standard errors for `/usr/bin/failure-event-handler` supervisor's subprocess.
- **supervisord.log:** captures the logs related to supervisor 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 the 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 |  
                ||     ||
```

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 the controller 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 the controller 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 automation controller 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 automation controller does not include the following rsyslog modules:

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

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

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

You can configure from the `/api/v2/settings/logging/` endpoint how the controller 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.

10.1 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 automation controller 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 controller 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 controller logs.

To configure various logging components in automation controller, click **Settings** from the left navigation bar then select **Logging settings** from the list of System options.

10.1.1 Log message schema

Common schema for all loggers:

- `cluster_host_id`: Unique identifier of the host within the controller 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 controller 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

10.1.2 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

10.1.3 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.

10.1.4 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

10.1.5 Job status changes

This is 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.

10.1.6 Controller 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 with the `ENABLE_EXTERNAL_LOGGING` option from the Logging settings page.

10.1.7 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

Automation controller'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://yourcontrollerfqdn.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_CONTROLLER_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:

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 in the **Logging Aggregator** field in the example below:



Sumologic

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

sumologic Library Search Metrics Dashboards Manage Help Alan (Re

Unnamed Search +

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

Last 15 Minutes Start
 Use Receipt Time

☆ Library | Save As | Info | Share | Live Tail | Report Slow Search

11/30/2016 2:58:21 PM -0500 11/30/2016 3:13:21 PM -0500

8
6
4
2

3:00 PM 3:05 PM 3:10 PM

2:58:21 PM STATUS: Done gathering results ELAPSED TIME: 00:00:01 RESULTS: 2 SESSION: 1B8BAE45AD7E172D 3:13:21 PM

Messages

Display Fields

- Time
- actor 1
- message2 2
- object1 2
- Message

Hidden Fields

- Collector 1
- Size
- Source 1
- Source Category 1
- Source Host 1
- Source Name 1

actor DISPLAY

VALUES	#	%
admin	2	100.00%

DRILLDOWN

Top Values Top Values Over Time Bottom Values

```
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 exampleszzzsafasdfqoqt\"]}",
level: "INFO",
@version: "1",
object2: "",
actor: "admin",
type: "Logstash"
}
```

Host: 207.67.11.130 Name: Http Input Category: Http Input

#	Time	Source	Message	View
2	11/30/2016 15:07:39.883-0500	admin	Activity Stream update entry for setting	View as Raw

```
{
  cluster_host_id: "tower",
  relationship: "",
  tags: [ ],
  @timestamp: "2016-11-30T20:07:39.883Z",
  object1: "setting",
```


Elastic stack (formerly ELK stack)

If starting from scratch, standing up your own version the elastic stack, the only change you required is to add the following lines to the logstash `logstash.conf` file:

```
filter {
  json {
    source => "message"
  }
}
```

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

10.2 Set Up Logging

To set up logging to any of the aggregator types:

1. Click **Settings** from the left navigation bar.
2. Under the list of System options, click to select **Logging settings**.
3. At the bottom of the Logging settings screen, click **Edit**.
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 TYPE ?

REVERT

Select types ▲

- logstash
- splunk
- loggly
- sumologic
- other

- **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 Aggregator 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

MISC. SYSTEM ACTIVITY STREAM **LOGGING**

ENABLE EXTERNAL LOGGING ?

LOGGING AGGREGATOR ? <input type="text" value="172.16.185.132"/> REVERT	LOGGING AGGREGATOR PORT ? <input type="text" value="80"/> REVERT	LOGGING AGGREGATOR TYPE ? <input type="text" value="splunk"/> REVERT
LOGGING AGGREGATOR USERNAME ? <input type="text"/>	LOGGING AGGREGATOR PASSWORD/TOKEN ? <input type="text" value="SHOW"/> REVERT	LOGGERS SENDING DATA TO LOG AGGREGATOR FORM ? <input type="text" value="awx, activity_stream, job_events"/> REVERT
LOG SYSTEM TRACKING FACTS INDIVIDUALLY ? <input type="checkbox"/>	LOGGING AGGREGATOR PROTOCOL ? <input type="text" value="HTTPS/HTTP"/> REVERT	* TCP CONNECTION TIMEOUT ? <input type="text" value="5"/> REVERT
LOGGING AGGREGATOR LEVEL THRESHOLD ? <input type="text" value="INFO"/> REVERT	ENABLE/DISABLE HTTPS CERTIFICATE VERIFICATION ? <input checked="" type="checkbox"/>	

REVERT ALL TO DEFAULT

TEST CANCEL SAVE

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 automation controller. 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.

10.3 Troubleshoot Logging

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.

METRICS

A metrics endpoint is available in the API: `/api/v2/metrics/` that surfaces instantaneous metrics about the controller, 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 controller node. Prometheus can be configured to scrape these metrics from the controller by hitting the controller 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.

11.1 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 controller user you have created, and a `<controller_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 the controller, the following example provides a valid scrape config if the URL for your controller's metrics endpoint was `https://controller_host:443/metrics`.

```
scrape_configs
- job_name: 'controller'
  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:
```

(continues on next page)

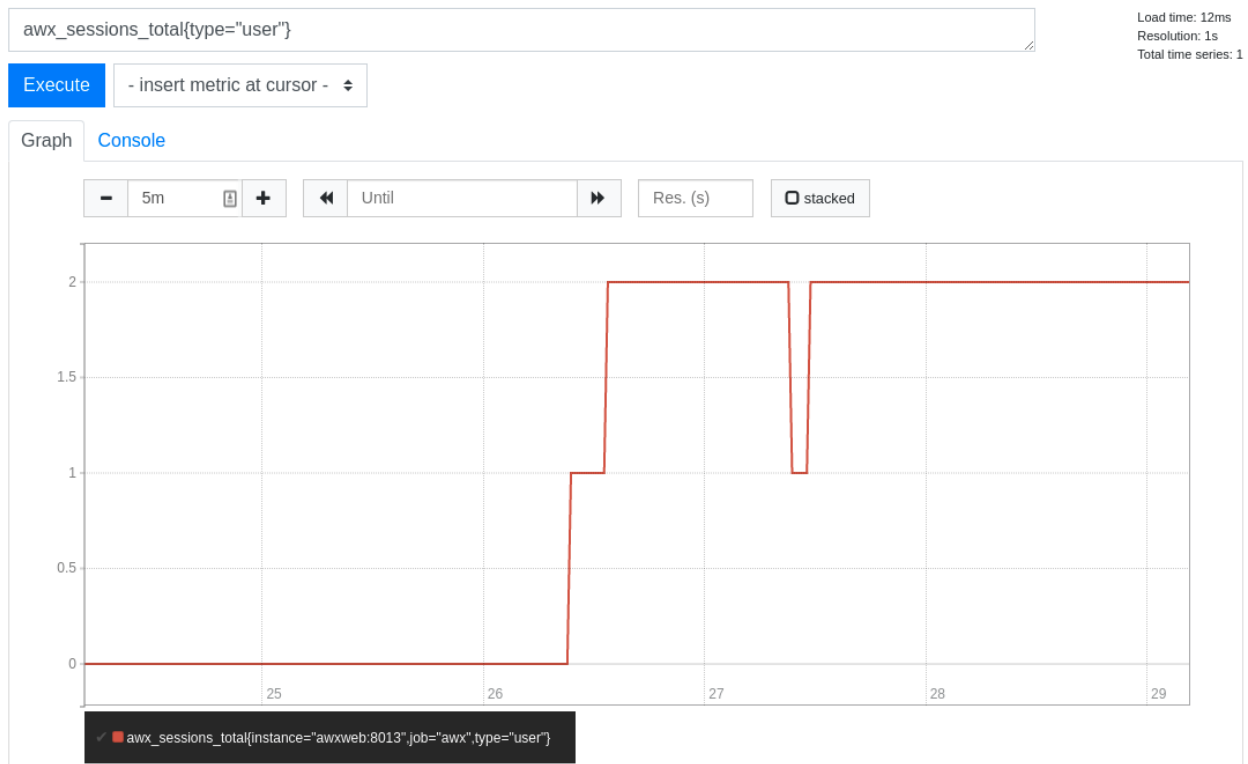
(continued from previous page)

```
- targets:
  - <controller_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 controller user sessions by executing: `awx_sessions_total{type="user"}`.



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

SECRET HANDLING AND CONNECTION SECURITY

This document describes how Red Hat Ansible Automation Platform controller handles secrets and connections in a secure fashion.

12.1 Secret Handling

automation controller manages three sets of secrets:

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

12.1.1 User passwords for local automation controller users

automation controller hashes local automation controller 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.

12.1.2 Secret handling for automation controller operational use

automation controller 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 automation controller 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 automation controller component services

These secrets are all stored unencrypted on the automation controller server, as they are all needed to be read by the automation controller service at startup in an automated fashion. All secrets are protected by Unix permissions, and restricted to root and the automation controller 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, the controller 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` the controller generated for you has been compromised and needs to be regenerated, you can run a tool from the installer that behaves much like the controller backup and restore tool. To generate a new secret key:

1. **Backup your controller database before you do anything else!** Follow the procedure described in the [Backing Up and Restoring](#) 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/`.

12.1.3 Secret handling for automation use

automation controller 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 automation controller settings
- “password” type survey fields entries

To encrypt secret fields, the controller 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, the controller fails to correctly decrypt the secret. automation controller is designed such that the `SECRET_KEY` is never readable in playbooks automation controller launches, that these secrets are never readable by the controller users, and no secret field values are ever made available via the automation controller 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.

12.2 Connection Security

12.2.1 Internal Services

automation controller 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.

12.2.2 External Access

automation controller 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.

12.2.3 Managed Nodes

automation controller 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).

SECURITY BEST PRACTICES

automation controller 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.

13.1 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

13.2 Understand the architecture of Ansible and the controller

Ansible and automation controller comprise a general purpose, declarative, automation platform. That means that once an Ansible playbook is launched (via the controller, 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 controller web UI, or the controller 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 the controller's API to perform automation.

We recommend all customers of automation controller select a secure default administrator password at time of installation. See *Changing the Controller Admin Password* for more information.

automation controller 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 automation controller on the open internet, significantly reducing the threat surface of your installation.

13.3 Granting access

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

- *Minimize administrative accounts*
- *Minimize local system access*
- *Remove access to credentials from users*
- *Enforce separation of duties*

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

13.3.2 Minimize local system access

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

13.3.3 Remove access to credentials from users

If an automation credential is only stored in the controller, 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.

13.3.4 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.

13.4 Available resources

Several resources exist in the controller 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*

13.4.1 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 automation controller, this is done via the built-in Activity Stream support that logs all changes within the controller, 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 automation controller be configured to use whatever IDS and/or logging/auditing (Splunk) is standard in your environment. automation controller includes built-in logging integrations for Elastic Stack, Splunk, Sumologic, Loggly, and more. See *Logging and Aggregation* for more information.

13.4.2 Existing security functionality

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

13.4.3 External account stores

Maintaining a full set of users just in the controller can be a time-consuming task in a large organization, prone to error. Automation controller 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.

13.4.4 Django password policies

Controller admins can leverage Django to set password policies at creation time via `AUTH_PASSWORD_VALIDATORS` to validate controller user passwords. In the `custom.py` file located at `/etc/tower/conf.d` of your controller 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 controller instance for the change to take effect. See *Starting, Stopping, and Restarting the Controller* for detail.

THE *AWX-MANAGE* UTILITY

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

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

14.1 Inventory Import

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

To use `awx-manage` properly, you must first create an inventory in the controller 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 controller 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 the controller 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.

14.2 Cleanup of old data

`awx-manage` has a variety of commands used to clean old data from the controller. The controller administrators can use the controller 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.

14.3 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.

14.4 Token and session management

automation controller supports the following commands for OAuth2 token management:

- `create_oauth2_token`
- `revoke_oauth2_tokens`
- `cleartokens`
- `expire_sessions`
- `clearsessions`

14.4.1 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: j89ia80079te6IAZ97L7E8bMgXCON2
```

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.

14.4.2 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 --revoke_refresh
```

To revoke all OAuth2 tokens for the user with `id=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 `id=example_user`:

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

14.4.3 cleartokens

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

14.4.4 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
```

14.4.5 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 controller user interface, see the [Applications](#) section of the *Automation Controller User Guide*.

14.5 Analytics gathering

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

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

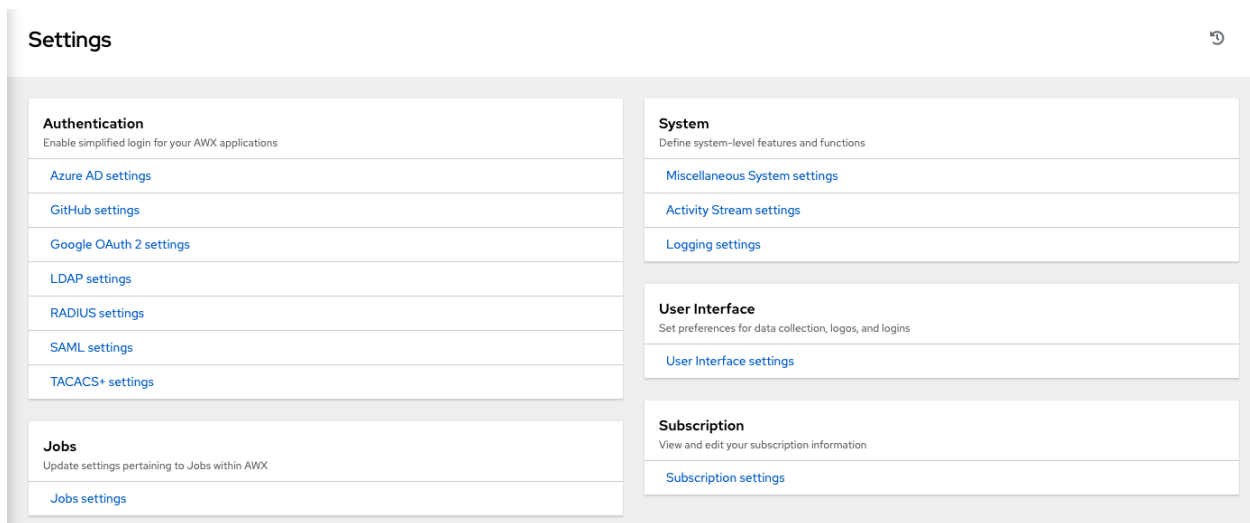
For customers with disconnected environments who want to collect usage information about unique hosts automated across a time period, use this command:

```
awx-manage host_metric --since YYYY-MM-DD --until YYYY-MM-DD --json
```

The parameters `--since` and `--until` specify date ranges and are optional, but one of them has to be present. The `--json` flag specifies the output format and is optional.

CONTROLLER CONFIGURATION

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



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 to their factory default values.

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

15.1 Authentication

Through the controller 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.

1. From the left navigation bar, click **Settings**.
2. The left side of the Settings window is a set of configurable Authentication settings. Select from the following options:
 - *Azure AD settings*
 - *GitHub settings*
 - *Google OAuth2 settings*
 - *LDAP settings*


- *RADIUS settings*
- *SAML settings*
- *TACACS+ settings*

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

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

15.2 Jobs

The Jobs tab allows you to configure the types of modules that are allowed to be used by the controller'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 the controller.

1. From the left navigation bar, click **Settings** from the left navigation bar and select **Jobs settings** 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 [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 REVERT

- 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 REVERT

* MAXIMUM SCHEDULED JOBS REVERT

PATHS TO EXPOSE TO ISOLATED JOBS REVERT

ANSIBLE CALLBACK PLUGINS REVERT

PATHS TO HIDE FROM ISOLATED JOBS REVERT

* ENABLE JOB ISOLATION REVERT

DEFAULT JOB TIMEOUT REVERT

DEFAULT INVENTORY UPDATE TIMEOUT REVERT

DEFAULT PROJECT UPDATE TIMEOUT REVERT

PER-HOST ANSIBLE FACT CACHE TIMEOUT REVERT

MAXIMUM NUMBER OF FORKS PER JOB REVERT

* RUN PROJECT UPDATES WITH HIGHER VERBOSITY REVERT

IGNORE ANSIBLE GALAXY SSL CERTIFICATE VERIFICATION REVERT

ENABLE ROLE DOWNLOAD REVERT

ENABLE COLLECTION(S) DOWNLOAD REVERT

FOLLOW SYMLINKS REVERT

ISOLATED HOST KEY CHECKING REVERT

* ISOLATED STATUS CHECK INTERVAL REVERT

* ISOLATED LAUNCH TIMEOUT REVERT

ISOLATED CONNECTION TIMEOUT REVERT

ENABLE DETAILED RESOURCE PROFILING ON ALL PLAYBOOK RUNS REVERT

EXTRA ENVIRONMENT VARIABLES REVERT

```
1 {}
```

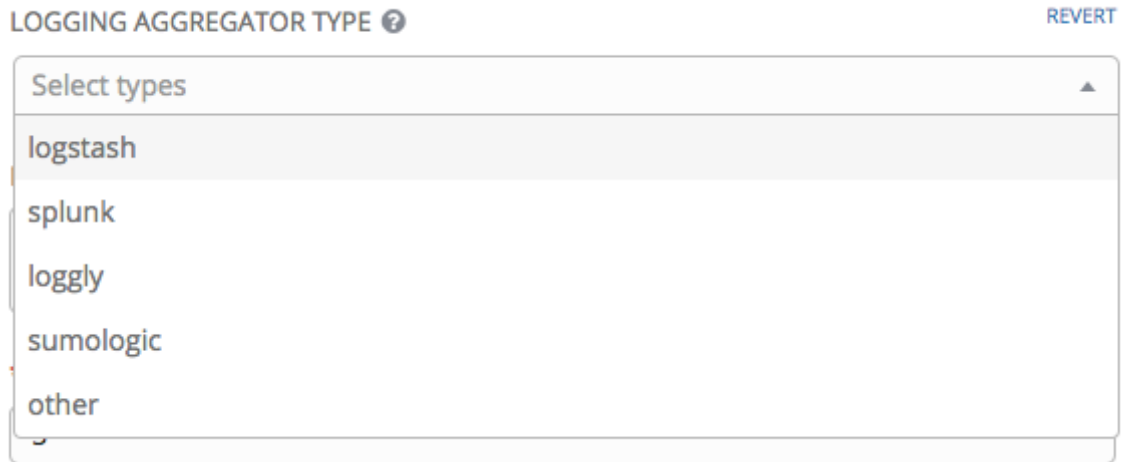
REVERT ALL TO DEFAULT CANCEL SAVE

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


15.3 System

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

1. From the left navigation bar, click **Settings**.
2. The right side of the Settings window is a set of configurable System settings. Select from the following options:
 - **Miscellaneous System settings:** define the base URL for the controller host, enable controller administration alerts, and allow all users to be visible to organization administrators.
 - **Activity Stream settings:** enable or disable activity stream.
 - **Logging settings:** configure logging options based on the type you choose:



For more information about each of the logging aggregation types, refer to the [Controller Logging and Aggregation](#) section of the *Automation Controller 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.

SETTINGS / SYSTEM

SYSTEM

MISC. SYSTEM ACTIVITY STREAM LOGGING

* BASE URL OF THE TOWER HOST REVERT

* IDLE TIME FORCE LOG OUT REVERT

ALLOW EXTERNAL USERS TO CREATE OAUTH2 TOKENS REVERT

REFRESH TOKEN EXPIRATION REVERT

CUSTOM VIRTUAL ENVIRONMENT PATHS REVERT

RED HAT CUSTOMER PASSWORD REVERT

* ALL USERS VISIBLE TO ORGANIZATION ADMINS REVERT

* ORGANIZATION ADMINS CAN MANAGE USERS AND TEAMS REVERT

* MAXIMUM NUMBER OF SIMULTANEOUS LOGGED IN SESSIONS REVERT

LOGIN REDIRECT OVERRIDE URL REVERT

AUTHORIZATION CODE EXPIRATION REVERT

GATHER DATA FOR AUTOMATION ANALYTICS REVERT

AUTOMATION ANALYTICS UPLOAD URL REVERT

* ENABLE HTTP BASIC AUTH REVERT

ACCESS TOKEN EXPIRATION REVERT

* REMOTE HOST HEADERS REVERT

RED HAT CUSTOMER USERNAME REVERT

AUTOMATION ANALYTICS GATHER INTERVAL REVERT

REVERT ALL TO DEFAULT CANCEL SAVE

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.

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

15.4 User Interface

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

Access the User Interface settings by clicking **Settings** from the left navigation bar and select **User Interface settings** from the Settings screen.

SETTINGS / USER INTERFACE

USER INTERFACE

* USER ANALYTICS TRACKING STATE REVERT

CUSTOM LOGO REVERT
 Choose file

CUSTOM LOGIN INFO REVERT

REVERT ALL TO DEFAULT CANCEL SAVE

15.4.1 Usability Analytics and Data Collection

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

Only users installing a trial of Red Hat Ansible Automation Platform or a fresh installation of automation controller 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 in the **User Interface settings**, by clicking **Settings** from the left navigation bar.

Automation controller collects user data automatically to help improve the product. You can control the way the controller collects data by setting your participation level in the **User Interface settings** 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>.

15.4.2 Custom Logos and Images

automation controller 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.

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  REVERT

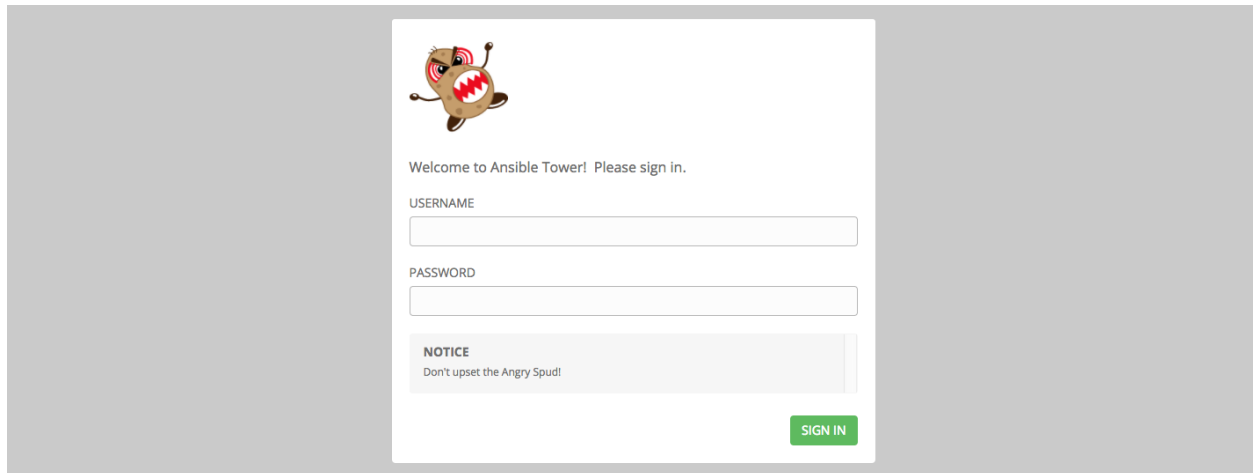
Off


CUSTOM LOGIN INFO  REVERT

Don't upset the Angry Spud!

[REVERT ALL TO DEFAULT](#)

The Tower login dialog would look like this:





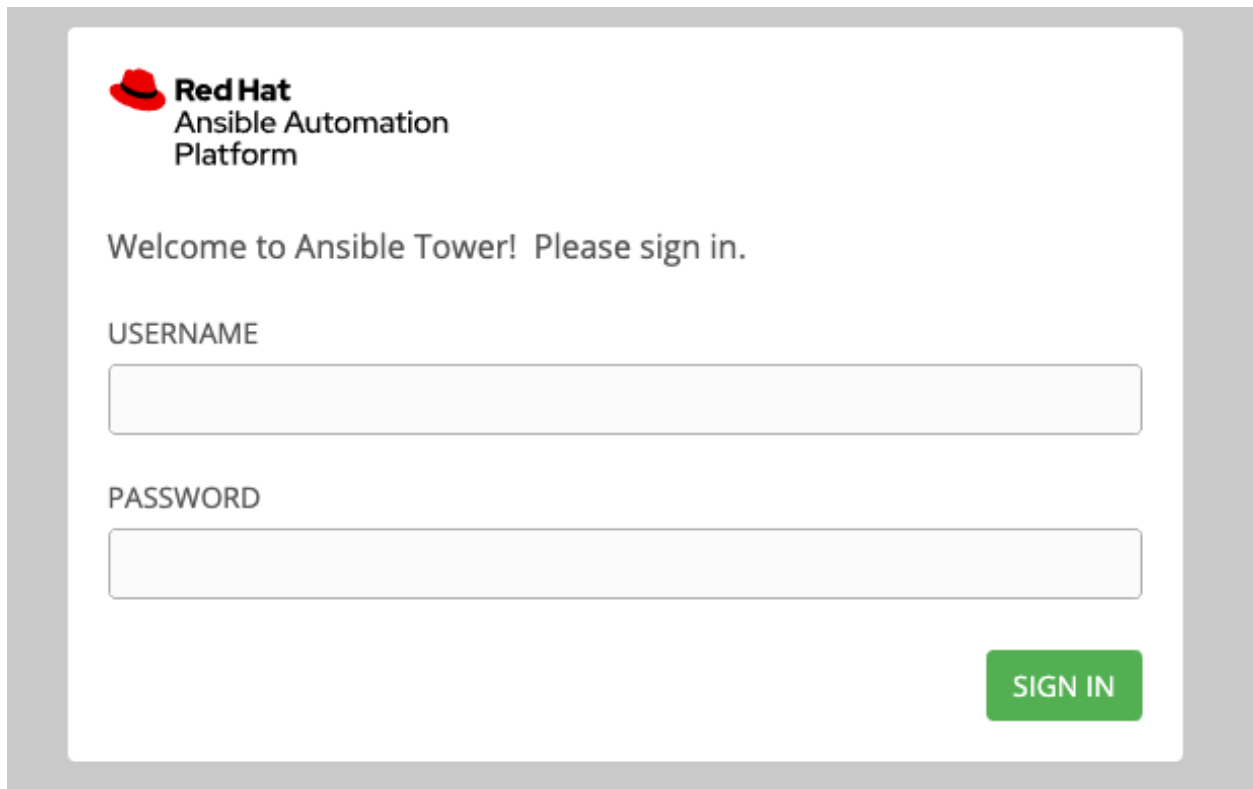
 Welcome to Ansible Tower! Please sign in.


USERNAME

PASSWORD

NOTICE
Don't upset the Angry Spud!

Selecting `Revert` will result in the appearance of the standard automation controller logo.




Red Hat
 Ansible Automation
 Platform

Welcome to Ansible Tower! Please sign in.

USERNAME

PASSWORD

15.5 License

Available subscriptions or a subscription manifest authorize the use of the automation controller. To obtain your automation controller 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](#) in the *Automation Controller User Guide* for more detail.

If you **have** a Red Hat Ansible Automation Platform subscription, use your Red Hat customer credentials when you launch the controller 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 controller 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 controller 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 automation controller 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 the controller launches for the first time, the Subscription screen automatically displays.

1. By default, the option to retrieve and import your subscription is to upload a subscription manifest you generate from https://access.redhat.com/management/subscription_allocations. See [Obtaining a subscriptions manifest](#) for more detail. Once you have a subscription manifest, you can upload it by browsing to the location where the file is saved (the subscription manifest is the complete .zip file, not its component parts).

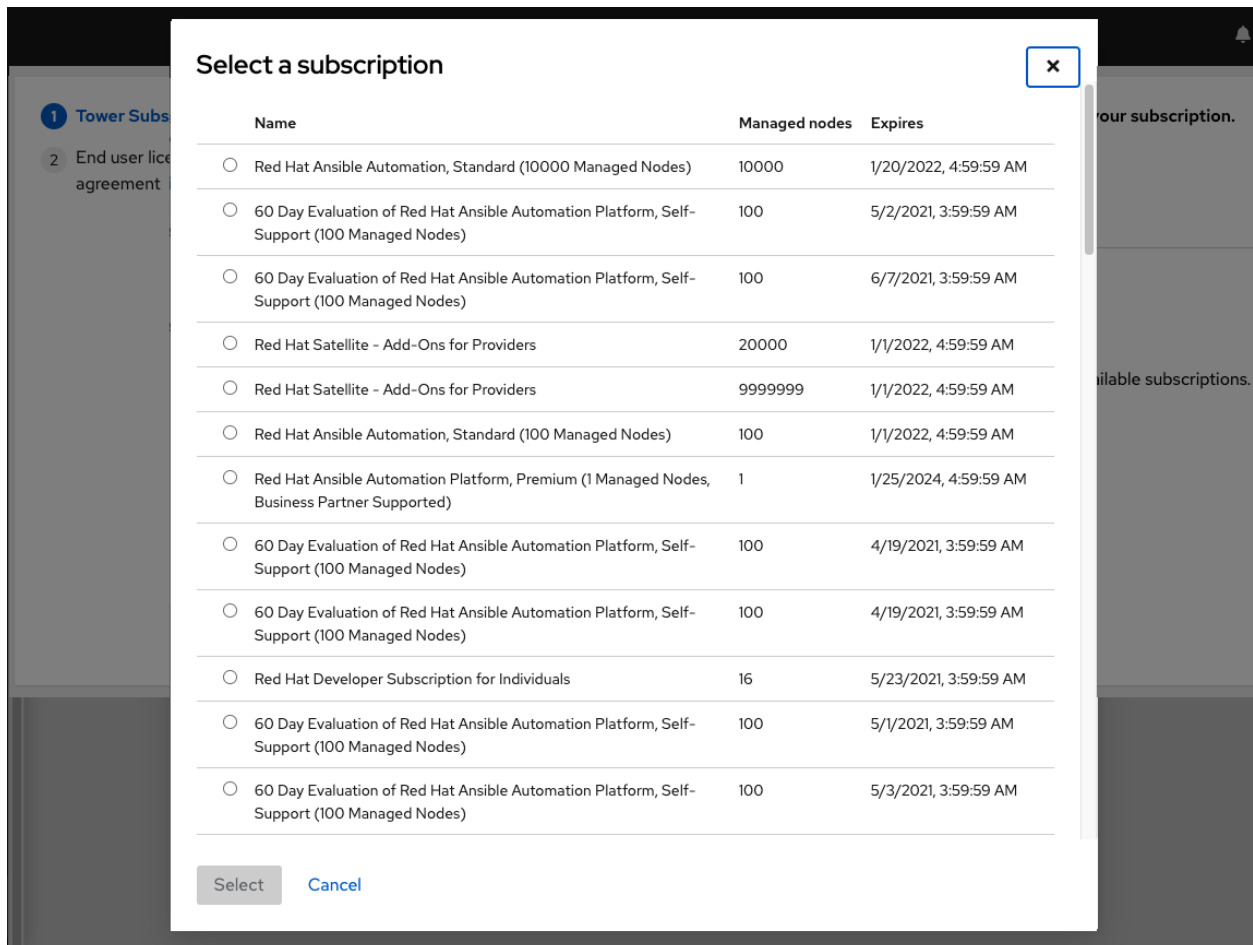
Note: If the **Browse** button in the subscription manifest option is grayed-out, clear the username and password fields to enable the **Browse** button.

Alternatively, you can choose the option to enter your Red Hat customer credentials using your username and password. Use your Satellite username/password if your controller cluster nodes are registered to Satellite via Subscription Manager. Once you entered your credentials, click **Get Subscriptions**.

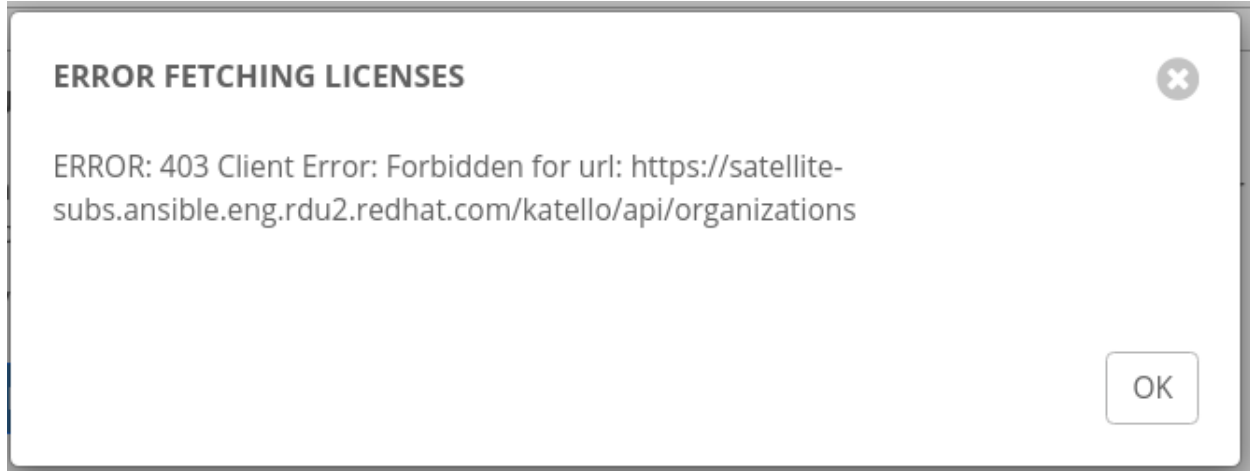
2. The subscription metadata is then retrieved from the RHSM/Satellite API, or from the manifest provided.

- If it is a subscription manifest, and multiple subscription counts were applied in a single installation, the controller will combine the counts but use the earliest expiration date as the expiry (at which point you will need to refresh your subscription).
- If you entered your credential information (username/password), the controller 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 the controller 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 in the Subscription details of the Subscription settings window), you will need to renew it in the controller by one of these two methods.



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



The Satellite username/password is used to query the Satellite API for existing subscriptions. From the Satellite API, the automation controller 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 controller 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. Click **Next** to proceed to the End User Agreement.
4. Review and check the **I agree to the End User License Agreement** checkbox and click **Submit**.

Once your subscription has been accepted, the controller briefly displays the subscription details and navigates you to the Dashboard of the automation controller interface. For later reference, you can return to this screen by clicking **Settings** from the left navigation bar and select **Subscription settings** from the Subscription option.

[Settings](#) > [Subscription](#)



Details

[← Back to Settings](#) **Subscription Details**

Status	✔ Compliant	Version	4.0.0	Subscription type	enterprise
Subscription	Red Hat Ansible Automation, Standard (10000 Managed Nodes)	Trial	False	Expires on	1/19/2022, 9:59:59 PM
Expires on UTC	1/20/2022, 4:59:59 AM	Days remaining	280	Hosts used	1
Hosts remaining	9999				

If you are ready to upgrade or renew, please [contact us](#).

[Edit](#)

ISOLATION FUNCTIONALITY AND VARIABLES

Automation controller uses container technology to isolate jobs from each other. By default, only the current project is exposed to the container running a job template.

You may find that you need to customize your playbook runs to expose additional directories. To fine tune your usage of job isolation, there are certain variables that can be set.

By default, automation controller 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 Jobs settings screen, or in the REST API at `/api/v2/settings/jobs`:

```
AWX_ISOLATION_BASE_PATH = "/opt/tmp"
```

If there are any additional directories that should specifically be exposed from the host to the container that playbooks run in, you can specify those in the **Paths to Expose to Isolated Jobs** field of the Jobs setting screen, or in the REST API at `/api/v2/settings/jobs`:

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

Note: The primary file you may want to add to `AWX_ISOLATION_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 REVERT

- 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 REVERT

/tmp

* MAXIMUM SCHEDULED JOBS REVERT

10

PATHS TO EXPOSE TO ISOLATED JOBS REVERT

ANSIBLE CALLBACK PLUGINS REVERT

PATHS TO HIDE FROM ISOLATED JOBS REVERT

* ENABLE JOB ISOLATION REVERT

DEFAULT JOB TIMEOUT REVERT

0

DEFAULT INVENTORY UPDATE TIMEOUT REVERT

0

DEFAULT PROJECT UPDATE TIMEOUT REVERT

0

PER-HOST ANSIBLE FACT CACHE TIMEOUT REVERT

0

MAXIMUM NUMBER OF FORKS PER JOB REVERT

200

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.

17.1 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":
↪ "7Ft7ym8MpE54yWGUNvxxg6KqGwPFsyhYn9QQfYHlgBxai74Qp1GE4zsvJduOfSFkTfWFfnPzYpxqcRsy1KacD0HH0vOAQUJDJC
↪ ",
"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.

17.1.1 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 controller 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.

17.1.2 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.

17.2 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_
↪controller CLI token", "application": null, "scope": "write"}' https://<USERNAME>:
↪<PASSWORD>@<CONTROLLER_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://controller/api/v2/job_templates/5/launch/
```

In automation controller, 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 controller user interface, as well as configure the expiration of an access token and its associated refresh token (if applicable).

SETTINGS / SYSTEM

SYSTEM

MISC. SYSTEM ACTIVITY STREAM LOGGING

* BASE URL OF THE TOWER HOST REVERT

* ALL USERS VISIBLE TO ORGANIZATION ADMINS REVERT

* ORGANIZATION ADMINS CAN MANAGE USERS AND TEAMS REVERT

* IDLE TIME FORCE LOG OUT REVERT

* MAXIMUM NUMBER OF SIMULTANEOUS LOGGED IN SESSIONS REVERT

* ENABLE HTTP BASIC AUTH REVERT

ALLOW EXTERNAL USERS TO CREATE OAUTH2 TOKENS REVERT

LOGIN REDIRECT OVERRIDE URL REVERT

ACCESS TOKEN EXPIRATION REVERT

REFRESH TOKEN EXPIRATION REVERT

AUTHORIZATION CODE EXPIRATION REVERT

* REMOTE HOST HEADERS REVERT

CUSTOM VIRTUAL ENVIRONMENT PATHS REVERT

GATHER DATA FOR AUTOMATION ANALYTICS REVERT

RED HAT CUSTOMER USERNAME REVERT

RED HAT CUSTOMER PASSWORD REVERT

AUTOMATION ANALYTICS UPLOAD URL REVERT

AUTOMATION ANALYTICS GATHER INTERVAL REVERT

[REVERT ALL TO DEFAULT](#) CANCEL SAVE

17.2.1 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 automation controller. 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://<controller>/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://<controller>/api/v2/tokens/<pk>/
```

Similarly, using a token:

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


17.3 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 the controller, 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 controller user interface. Refer to the [Applications](#) section of the *Automation Controller User Guide* for more detail.

17.3.1 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 automation controller, that webapp may need to create OAuth2 Tokens on behalf of users in that other webapp. Creating an application in the controller with the `authorization code` grant type is the preferred way to do this because:

- this allows an external application to obtain a token from the controller 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://<controller>/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. The controller 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 the controller with the `code`, `client_id`, `client_secret`, `grant_type`, and `redirect_uri`.
3. The controller 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 **System settings** screen:

SETTINGS / SYSTEM

SYSTEM

MISC. SYSTEM ACTIVITY STREAM LOGGING

* BASE URL OF THE TOWER HOST REVERT

* ALL USERS VISIBLE TO ORGANIZATION ADMINS REVERT

* ORGANIZATION ADMINS CAN MANAGE USERS AND TEAMS REVERT

* IDLE TIME FORCE LOG OUT REVERT

* MAXIMUM NUMBER OF SIMULTANEOUS LOGGED IN SESSIONS REVERT

* ENABLE HTTP BASIC AUTH REVERT

ALLOW EXTERNAL USERS TO CREATE OAUTH2 TOKENS

LOGIN REDIRECT OVERRIDE URL REVERT

ACCESS TOKEN EXPIRATION REVERT

REFRESH TOKEN EXPIRATION REVERT

AUTHORIZATION CODE EXPIRATION REVERT

* REMOTE HOST HEADERS REVERT

CUSTOM VIRTUAL ENVIRONMENT PATHS REVERT

GATHER DATA FOR AUTOMATION ANALYTICS REVERT

RED HAT CUSTOMER USERNAME REVERT

RED HAT CUSTOMER PASSWORD REVERT

AUTOMATION ANALYTICS UPLOAD URL REVERT

AUTOMATION ANALYTICS GATHER INTERVAL REVERT

REVERT ALL TO DEFAULT

CANCEL SAVE

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 automation controller 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 the controller, for which you want to provide access tokens. To do this, have your Administrator add this whitelist to their local controller settings:

```
CORS_ALLOWED_ORIGIN_REGEXES = [
    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 the controller.

17.3.2 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",
}
```

(continues on next page)

(continued from previous page)

```

    "user": 1,
    "client_id": "gwSPoasWSdNkMDtBN3Hu2WYQpPWC09SwUEsKK221",
    "client_secret":
↵ "fI6ZpfocHYBGfmltP92r0yIgCyfRdQt0Tos9L8a4fNsJjQQMwp9569eIaUBsaVDgt2eiwOGe0bg5m5vCSstC1Zmtdy359RVx
↵ ",
    "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://<controller>/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"}

```

17.4 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.

17.4.1 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
↪ "gwSPoasWSdNkMdBN3Hu2WYQpPWCO9SwUEsKK22l:fI6ZpfocHYBGfmltP92r0yIgCyfRdDQt0Tos9L8a4fNsJjQQMwp9569e"
↪ " \
  http://<controller>/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": "NDInWxGJI4iZgqpsreuujbvzCfJqgR", "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.

17.4.2 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
↪ "gwSPoasWSdNkMDtBN3Hu2WYQpPWC09SwUEsKK22l:fI6ZpfocHYBGfm1tP92r0yIgCyfRdDQt0Tos9L8a4fNsJjQQMwp9569e:
↪ " \
http://<controller>/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 [Token and session management](#) section.

This setting can be configured at the system-level in the automation controller User Interface:

SETTINGS / SYSTEM

SYSTEM

MISC. SYSTEM ACTIVITY STREAM LOGGING

* BASE URL OF THE TOWER HOST <small>REVERT</small> <input type="text" value="https://ec2-3-89-219-44.compute-1.amazonaws.com"/>	* ALL USERS VISIBLE TO ORGANIZATION ADMINS <small>REVERT</small> <input checked="" type="checkbox"/>	* ORGANIZATION ADMINS CAN MANAGE USERS AND TEAMS <small>REVERT</small> <input checked="" type="checkbox"/>
* IDLE TIME FORCE LOG OUT <small>REVERT</small> <input type="text" value="36000"/>	* MAXIMUM NUMBER OF SIMULTANEOUS LOGGED IN SESSIONS <small>REVERT</small> <input type="text" value="-1"/>	* ENABLE HTTP BASIC AUTH <small>REVERT</small> <input checked="" type="checkbox"/>
ALLOW EXTERNAL USERS TO CREATE OAUTH2 TOKENS <small>REVERT</small> <input type="checkbox"/>	LOGIN REDIRECT OVERRIDE URL <small>REVERT</small> <input type="text"/>	ACCESS TOKEN EXPIRATION <small>REVERT</small> <input type="text" value="3153600000"/>
REFRESH TOKEN EXPIRATION <small>REVERT</small> <input type="text" value="2628000"/>	AUTHORIZATION CODE EXPIRATION <small>REVERT</small> <input type="text" value="600"/>	* REMOTE HOST HEADERS <small>REVERT</small> <input type="text" value="REMOTE_ADDR, REMOTE_HOST"/>
CUSTOM VIRTUAL ENVIRONMENT PATHS <small>REVERT</small> <input type="text"/>	GATHER DATA FOR AUTOMATION ANALYTICS <small>REVERT</small> <input checked="" type="checkbox"/>	RED HAT CUSTOMER USERNAME <small>REVERT</small> <input type="text"/>
RED HAT CUSTOMER PASSWORD <small>REVERT</small> <input type="text" value="SHOW"/>	AUTOMATION ANALYTICS UPLOAD URL <small>REVERT</small> <input type="text" value="https://cloud.redhat.com/api/ingress/v1/upload"/>	AUTOMATION ANALYTICS GATHER INTERVAL <small>REVERT</small> <input type="text" value="14400"/>

REVERT ALL TO DEFAULT

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

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 automation controller User Interface and saved to the PostgreSQL database. For instructions, refer to the *Controller Configuration* section.

Account authentication in automation controller 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.

18.1 GitHub settings

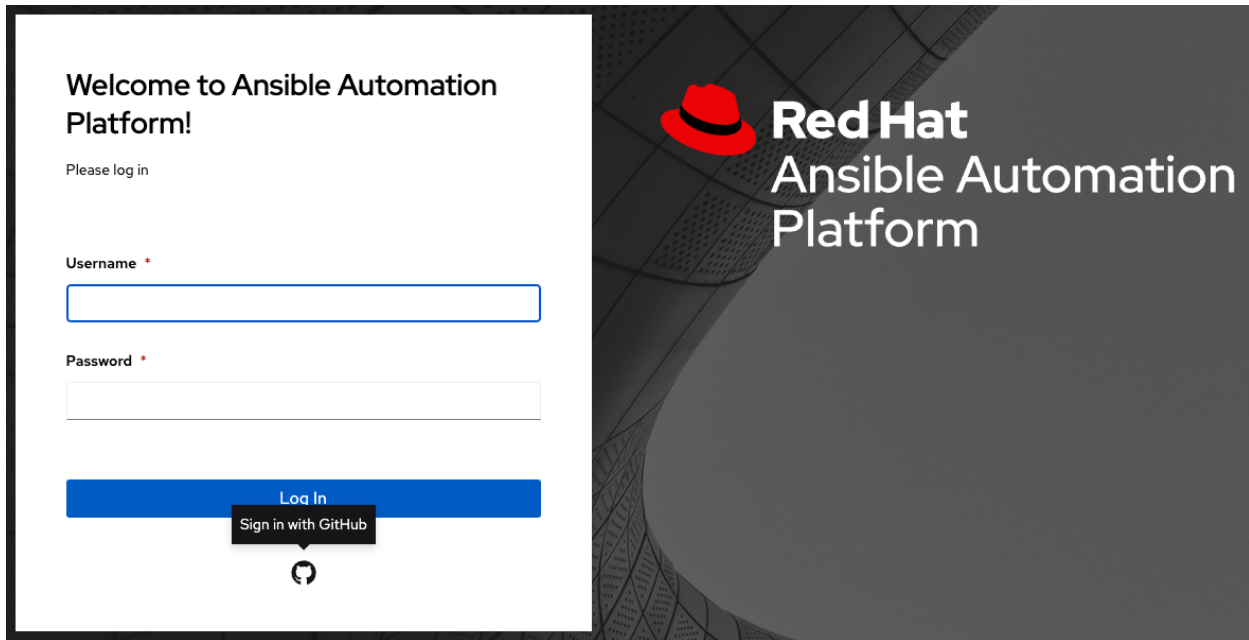
To set up social authentication for GitHub, you will need to obtain an OAuth2 key and secret for a web application. To do this, you must first register the new application with GitHub at <https://github.com/settings/developers>. In order to register the application, you must supply it with your homepage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

1. Click **Settings** from the left navigation bar.
2. On the left side of the Settings window, click **GitHub settings** from the list of Authentication options.
3. Click the **GitHub Default** tab if not already selected.

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. Click **Edit** and 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.

- To verify that the authentication was configured correctly, logout of automation controller and the login screen will now display the GitHub logo to allow logging in with those credentials.



18.1.1 GitHub Organization 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 the controller 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 Details page. 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 automation controller User Interface.

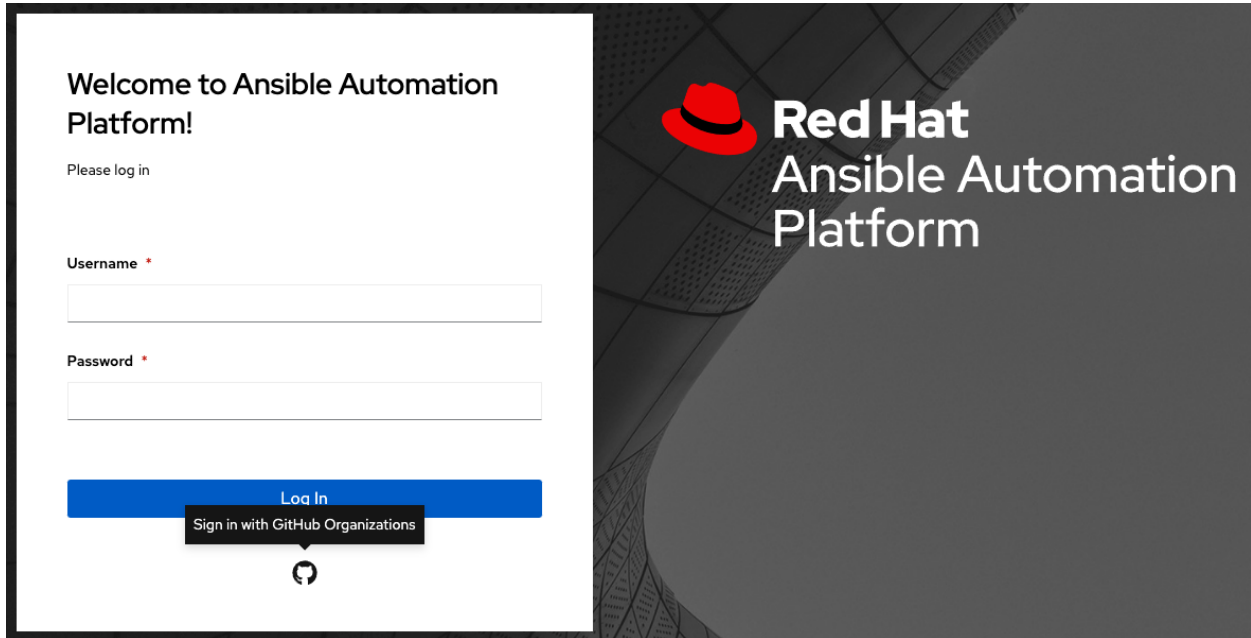
- Click **Settings** from the left navigation bar.
- On the left side of the Settings window, click **GitHub settings** from the list of Authentication options.
- Click the **GitHub Organization** tab.

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.

- Click **Edit** and copy and paste GitHub's Client ID into the **GitHub Organization OAuth2 Key** field.
- Copy and paste GitHub's Client Secret into the **GitHub Organization OAuth2 Secret** field.
- Enter the name of your GitHub organization, as used in your organization's URL (e.g., <https://github.com/<yourorg>>) in the **GitHub Organization Name** 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 automation controller and the login screen will now display the GitHub Organization logo to allow logging in with those credentials.



18.1.2 GitHub Team settings

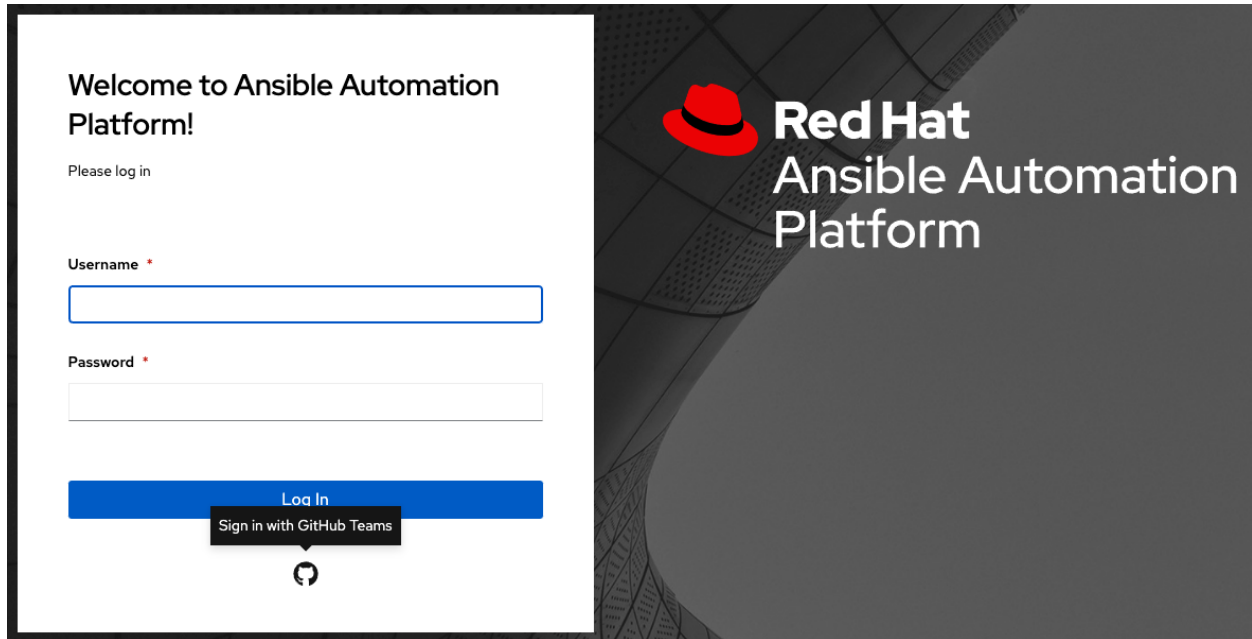
To set up social authentication for a GitHub Team, you will need to obtain an OAuth2 key and secret for a web application. To do this, you must first register your team-owned application at <https://github.com/organizations/<yourorg>/settings/applications>. In order to register the application, you must supply it with your Authorization callback URL, which is the **Callback URL** shown in the Details page. 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 automation controller 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 automation controller User Interface.
2. Click **Settings** from the left navigation bar.
3. On the left side of the Settings window, click **GitHub settings** from the list of Authentication options.
4. Click the **GitHub Team** tab.

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. Click **Edit** and 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. Copy and paste GitHub's team ID in the **GitHub Team ID** field.
8. For details on completing the mapping fields, see *Organization and Team Mapping*.

9. Click **Save** when done.
10. To verify that the authentication was configured correctly, logout of automation controller and the login screen will now display the GitHub Team logo to allow logging in with those credentials.



18.1.3 GitHub Enterprise settings

To set up social authentication for a GitHub Enterprise, you will need to obtain a GitHub Enterprise URL, an API URL, OAuth2 key and secret for a web application. To obtain the URLs, refer to the GitHub documentation on [GitHub Enterprise administration](#). To obtain the key and secret, you must first register your enterprise-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 Details page. Because its hosted on site and not github.com, you must specify which auth adapter it will talk to.

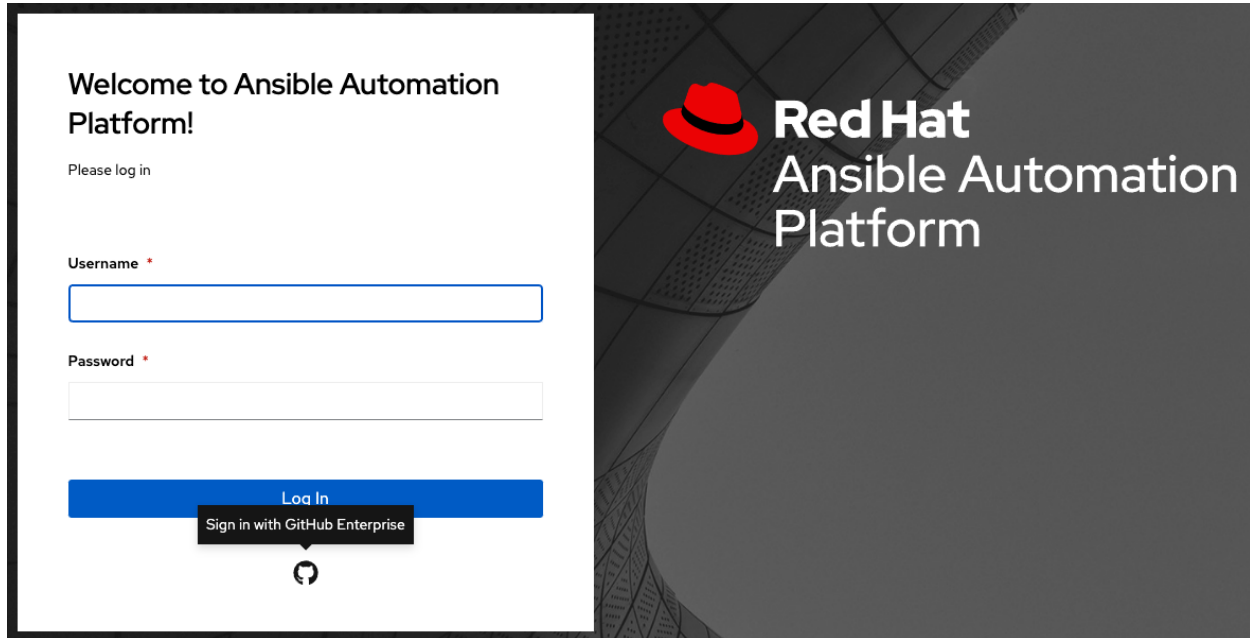
Each key and secret must belong to a unique application and cannot be shared or reused between different authentication backends. The OAuth2 key (Client ID) and secret (Client Secret) will be used to supply the required fields in the automation controller User Interface.

1. Click **Settings** from the left navigation bar.
2. On the left side of the Settings window, click **GitHub settings** from the list of Authentication options.
3. Click the **GitHub Enterprise** tab.

The **GitHub Enterprise 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. Click **Edit** to configure GitHub Enterprise settings.
5. In the **GitHub Enterprise URL** field, enter the hostname of the GitHub Enterprise instance (e.g., <https://github.example.com>).
6. In the **GitHub Enterprise API URL** field, enter the API URL of the GitHub Enterprise instance (e.g., <https://github.example.com/api/v3>).
7. Copy and paste GitHub's Client ID into the **GitHub Enterprise OAuth2 Key** field.
8. Copy and paste GitHub's Client Secret into the **GitHub Enterprise OAuth2 Secret** field.

9. For details on completing the mapping fields, see *Organization and Team Mapping*.
10. Click **Save** when done.
11. To verify that the authentication was configured correctly, logout of automation controller and the login screen will now display the GitHub Enterprise logo to allow logging in with those credentials.



18.1.4 GitHub Enterprise Organization settings

To set up social authentication for a GitHub Enterprise Org, you will need to obtain a GitHub Enterprise Org URL, an Org API URL, an Org OAuth2 key and secret for a web application. To obtain the URLs, refer to the GitHub documentation on [GitHub Enterprise administration](#). To obtain the key and secret, you must first register your enterprise 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 Details page. Because its hosted on site and not github.com, you must specify which auth adapter it will talk to.

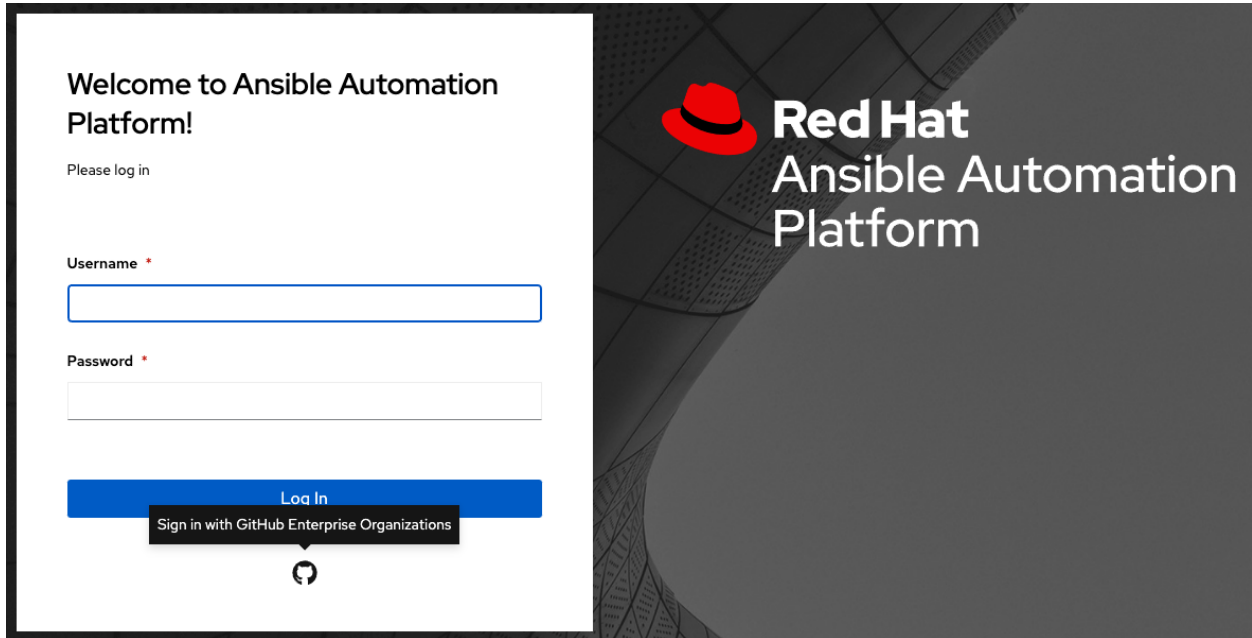
Each key and secret must belong to a unique application and cannot be shared or reused between different authentication backends. The OAuth2 key (Client ID) and secret (Client Secret) will be used to supply the required fields in the automation controller User Interface.

1. Click **Settings** from the left navigation bar.
2. On the left side of the Settings window, click **GitHub settings** from the list of Authentication options.
3. Click the **GitHub Enterprise Organization** tab.

The **GitHub Enterprise 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. Click **Edit** to configure GitHub Enterprise Organization settings.
5. In the **GitHub Enterprise Organization URL** field, enter the hostname of the GitHub Enterprise Org instance (e.g., <https://github.orgexample.com>).
6. In the **GitHub Enterprise Organization API URL** field, enter the API URL of the GitHub Enterprise Org instance (e.g., <https://github.orgexample.com/api/v3>)

7. Copy and paste GitHub's Client ID into the **GitHub Enterprise Organization OAuth2 Key** field.
8. Copy and paste GitHub's Client Secret into the **GitHub Enterprise Organization OAuth2 Secret** field.
9. Enter the name of your GitHub Enterprise organization, as used in your organization's URL (e.g., <https://github.com/<yourorg>>) in the **GitHub Enterprise Organization Name** field.
10. For details on completing the mapping fields, see *Organization and Team Mapping*.
11. Click **Save** when done.
12. To verify that the authentication was configured correctly, logout of automation controller and the login screen will now display the GitHub Enterprise Organization logo to allow logging in with those credentials.



18.1.5 GitHub Enterprise Team settings

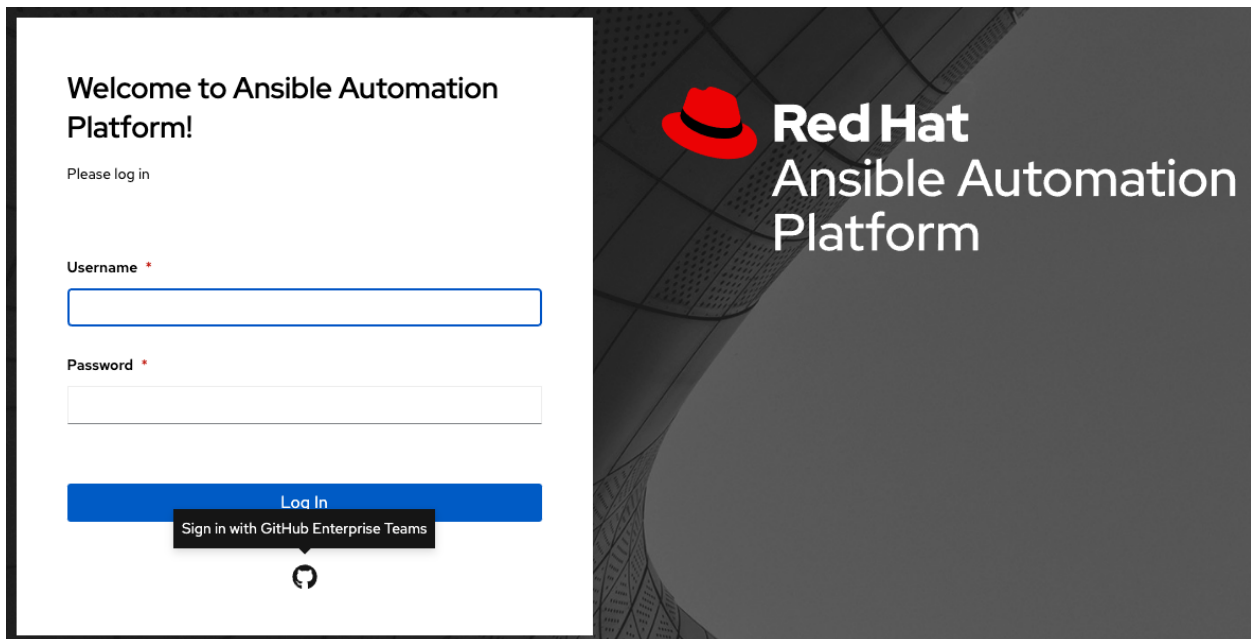
To set up social authentication for a GitHub Enterprise teams, you will need to obtain a GitHub Enterprise Org URL, an Org API URL, an Org OAuth2 key and secret for a web application. To obtain the URLs, refer to the GitHub documentation on [GitHub Enterprise administration](#). To obtain the key and secret, you must first register your enterprise 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 Details page. Because its hosted on site and not github.com, you must specify which auth adapter it will talk to.

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 automation controller 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 automation controller User Interface.
2. Click **Settings** from the left navigation bar.
3. On the left side of the Settings window, click **GitHub settings** from the list of Authentication options.
4. Click the **GitHub Enterprise Team** tab.

The **GitHub Enterprise 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. Click **Edit** to configure GitHub Enterprise Team settings.
6. In the **GitHub Enterprise Team URL** field, enter the hostname of the GitHub Enterprise team instance (e.g., <https://github.teamexample.com>).
7. In the **GitHub Enterprise Team API URL** field, enter the API URL of the GitHub Enterprise team instance (e.g., <https://github.teamexample.com/api/v3>)
8. Copy and paste GitHub's Client ID into the **GitHub Enterprise Team OAuth2 Key** field.
9. Copy and paste GitHub's Client Secret into the **GitHub Enterprise Team OAuth2 Secret** field.
10. Copy and paste GitHub's team ID in the **GitHub Enterprise Team ID** field.
11. For details on completing the mapping fields, see *Organization and Team Mapping*.
12. Click **Save** when done.
13. To verify that the authentication was configured correctly, logout of automation controller and the login screen will now display the GitHub Enterprise Teams logo to allow logging in with those credentials.



18.2 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 automation controller User Interface.

1. Click **Settings** from the left navigation bar.
2. On the left side of the Settings window, click **Google OAuth 2 settings** from the list of Authentication options.

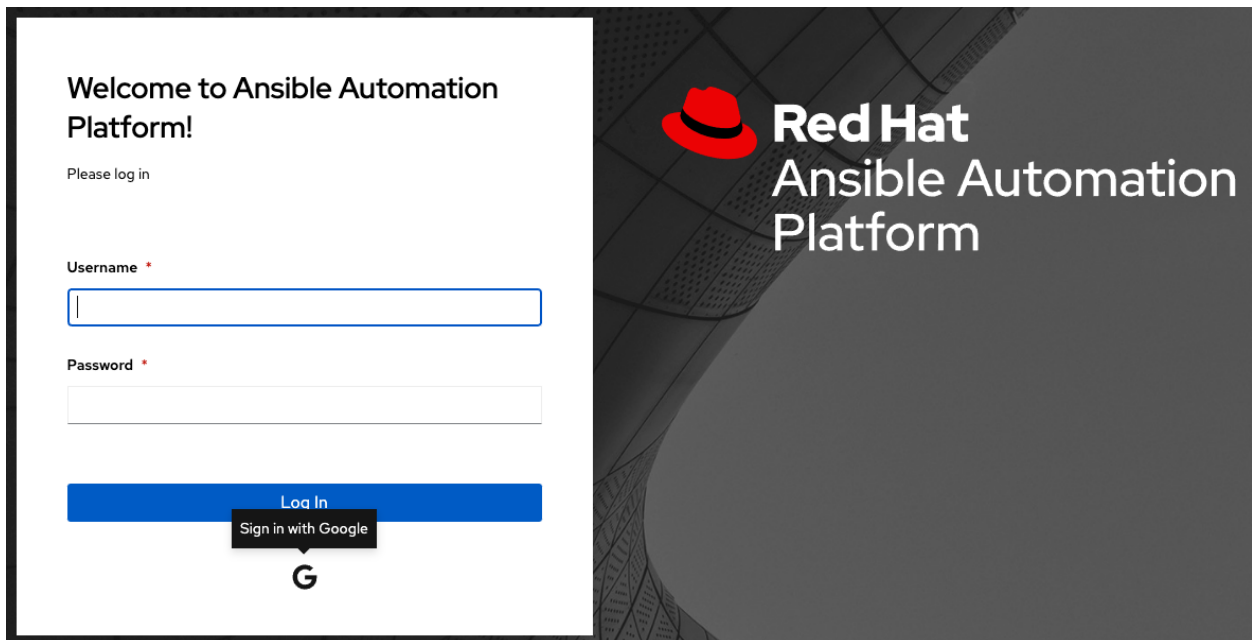
The **Google OAuth2 Callback URL** field is already pre-populated and non-editable.

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:

- Click **Edit** and 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.

The screenshot shows the 'Edit Details' page for Google OAuth2 configuration. The 'Google OAuth2 Key' field contains the value '528620852399-gm2dt4hrl2tsj67fqamk09kle0ad6gd...' and the 'Google OAuth2 Secret' field contains 'kSjrPnmKLQgdRmnMj8GHcDzy'. Both fields are highlighted with a red box. Below these are sections for 'Google OAuth2 Allowed Domains' and 'Google OAuth2 Extra Arguments', each with a 'Revert' button and a list of entries.

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 automation controller and the login screen will now display the Google logo to indicate it as a alternate method of logging into automation controller.



18.3 Organization and Team Mapping

18.3.1 Organization mapping

You will need to control which users are placed into which controller 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", "/^controller-[^@]+?@.*$/i"],
    "users": "/^[^@].*?@example\\.com$/ "
  }
}
```

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 = {}
```

18.3.2 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 the controller 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 = []
```


SETTING UP ENTERPRISE AUTHENTICATION

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

- *Azure AD settings*
- *LDAP Authentication*
- *RADIUS settings*
- *SAML settings*
 - *Transparent SAML Logins*
- *TACACS+ settings*

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:

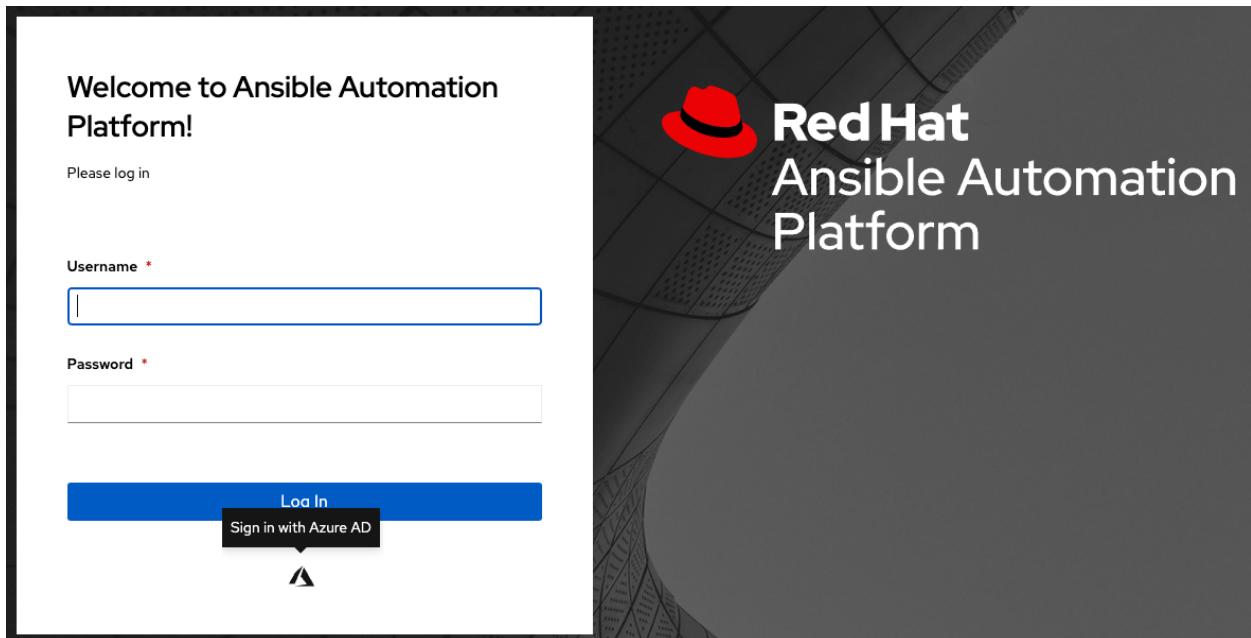
- Enterprise users can only be created via the first successful login attempt from remote authentication backend.
- Enterprise users cannot be created/authenticated if non-enterprise users with the same name has already been created in the controller.
- The controller passwords of enterprise users should 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 controller user by setting the password field. However, this operation is irreversible, as the converted controller user can no longer be treated as enterprise user.

19.1 Azure AD settings

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 Settings Authentication screen.

1. Click **Settings** from the left navigation bar.
2. On the left side of the Settings window, click **Azure AD settings** from the list of Authentication options.

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. Click **Edit** and 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 Settings - 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 automation controller and the login screen will now display the Microsoft Azure logo to allow logging in with those credentials.



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

19.2 LDAP Authentication

Refer to the *Setting up LDAP Authentication* section.

19.3 RADIUS settings

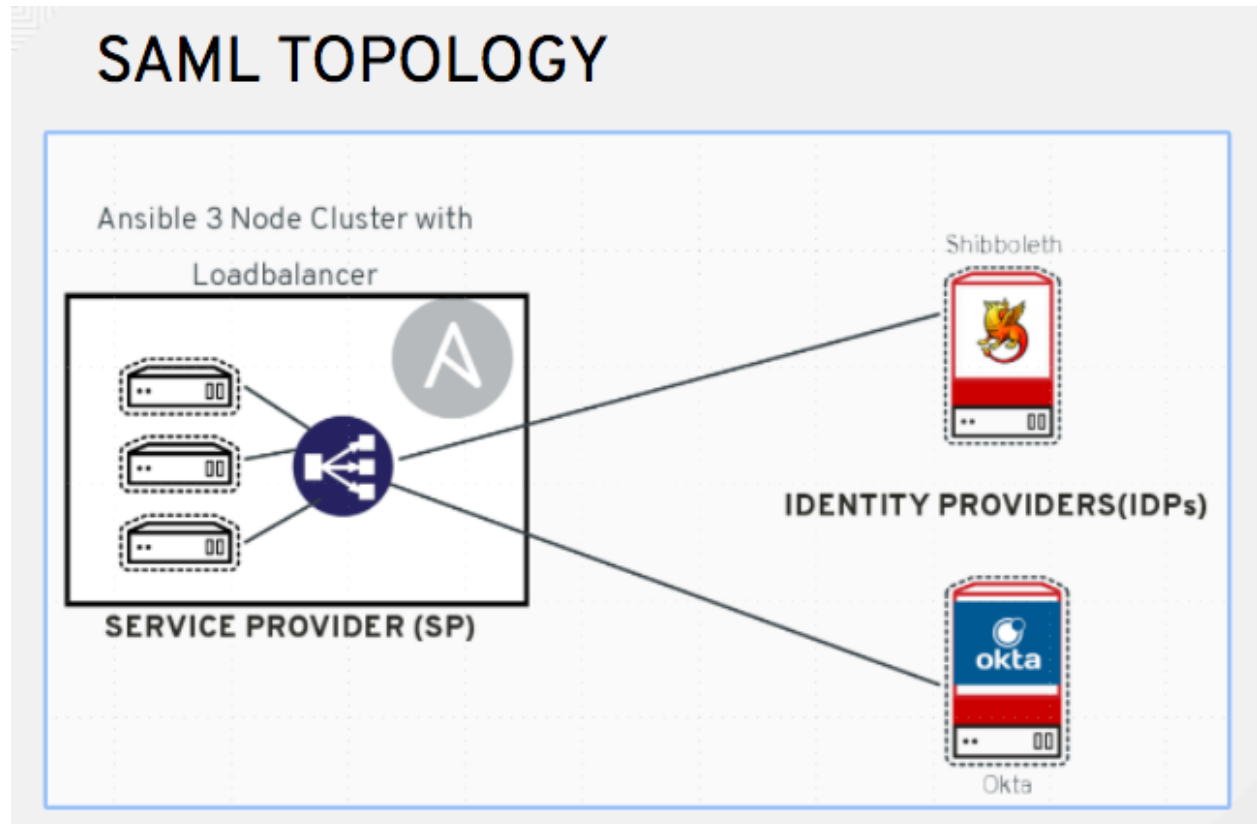
automation controller can be configured to centrally use RADIUS as a source for authentication information.

1. Click **Settings** from the left navigation bar.
2. On the left side of the Settings window, click **RADIUS settings** from the list of Authentication options.
3. Click **Edit** and 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.

19.4 SAML 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, automation controller). automation controller can be configured to talk with SAML in order to authenticate (create/login/logout) controller users. User Team and Organization membership can be embedded in the SAML response to the controller.



The following instructions describe automation controller as the service provider. To authenticate users through RHSSO (keycloak), refer to the [Red Hat Single Sign On Integration with the Automation Controller](#) blog.

To setup SAML authentication:

1. Click **Settings** from the left navigation bar.
2. On the left side of the Settings window, click **SAML settings** from the list of Authentication options.
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. Click **Edit** and set the **SAML Service Provider Entity ID** to be the same as the **Base URL of the controller host** field that can be found in the Miscellaneous System settings screen by clicking **Settings** from the left navigation bar. Through the API, it can be viewed in the `/api/v2/settings/system`, under the `CONTROLLER_BASE_URL` variable. The Entity ID can be set to any one of the individual controller 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 Base URL is different for each node in a cluster. Commonly, a load balancer will sit in front of many controller cluster nodes to provide a single entry point, the controller Cluster FQDN. The SAML Service Provider must be able establish an outbound connection and route to the controller Cluster Node or the controller Cluster FQDN set in the SAML Service Provider Entity ID.

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

5. Create a server certificate for the Ansible cluster. Typically when an Ansible cluster is configured, the controller 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 controller Cluster Nodes. SSL can either be enabled or disabled per individual controller 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.

As an example for public certs:

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

6. Create an optional private key for the controller 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 controller 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:



```
1 {
2   "en-US": {
3     "url": "https://ansible-tower-fqdn-elb.amazonaws.com",
4     "displayname": "Ansible Tower Cluster",
5     "name": "ansible-tower-cluster"
6   }
7 }
```

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

- 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:



```
1 {
2   "givenName": "Central IT",
3   "emailAddress": "central-it@company.com"
4 }
```

- 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 ⓘ
1 {
2   "givenName": "Central IT",
3   "emailAddress": "central-it@company.com"
4 }
```

10. In the **SAML Enabled Identity Providers** field, provide information on how to connect to each Identity Provider listed. The controller 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 controller 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 the controller and it generates a unique URL.
- `url` - the Single Sign On (SSO) URL the controller 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 (the controller 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"
  }
}
```

```

1 {
2   "myidp": {
3     "entity_id": "https://idp.example.com",
4     "url": "https://myidp.example.com/sso",
5     "x509cert": "MIIEJjCCAwGgAwIBAgIUFuSD540PSBhdDhh3gZorvrIaoAwDQYJKoZIhvcNAQEF\nbQAwXTElMkUeFmB0GA
6   },
7   "oneLogin": {
8     "entity_id": "https://app.oneLogin.com/saml/metadata/123456",
9     "url": "https://example.oneLogin.com/trust/saml2/http-post/sso/123456",
10    "x509cert": "MIIEJjCCAwGgAwIBAgIUFuSD540PSBhdDhh3gZorvrIaoAwDQYJKoZIhvcNAQEF\nbQAwXTElMkUeFmB0GA
11    "attr_user_permanent_id": "name_id",
12    "attr_first_name": "User.FirstName",
13    "attr_last_name": "User.LastName",
14    "attr_username": "User.email",
15    "attr_email": "User.email"
16  }
17 }

```

Warning: Do not create a SAML user that shares the same email with another user (including a non-SAML user). Doing so will result in the accounts being 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 the controller is unable to read. Instead, the controller 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"
  }
}

```

- Optionally provide the **SAML Organization Map**. For further detail, see *Organization and Team Mapping*.
- The controller can be configured to look for particular attributes that contain Team and Organization membership to associate with users when they log into the controller. 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"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">

```

(continues on next page)

(continued from previous page)

```
<saml2:AttributeValue>Engineering</saml2:AttributeValue>
</saml2:Attribute>
</saml2:AttributeStatement>
```

Below is the corresponding controller 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 controller Team -> controller Organization. This is needed because the same named Team can exist in multiple Organizations in the controller. 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"
}
```

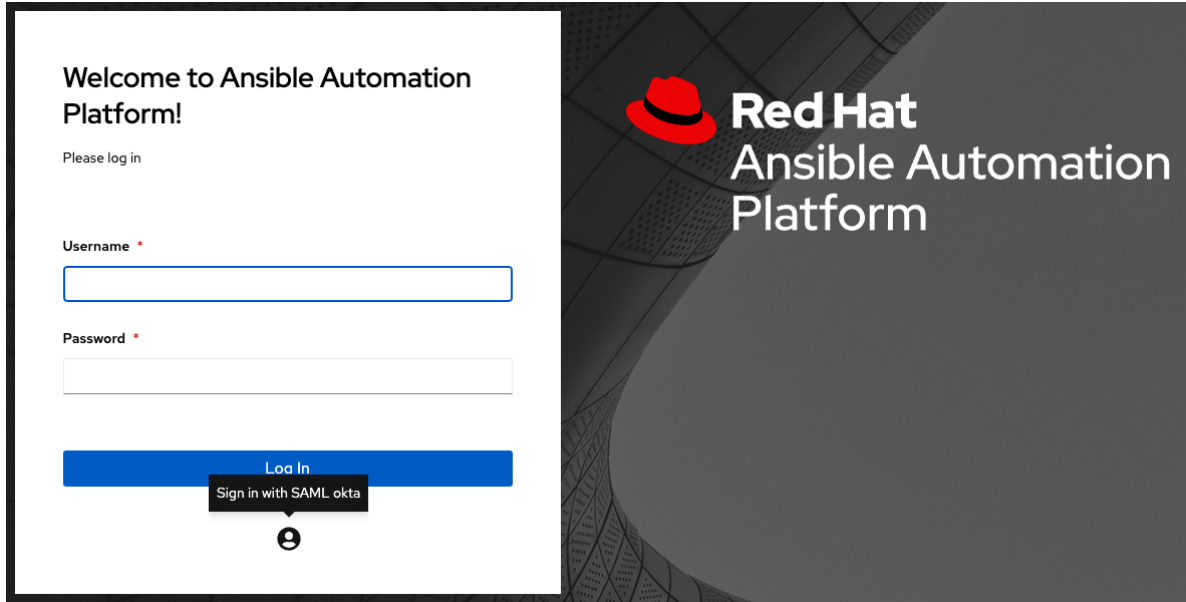
Once the user authenticates, the controller creates organization and team aliases, as expected.

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.

The controller 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.

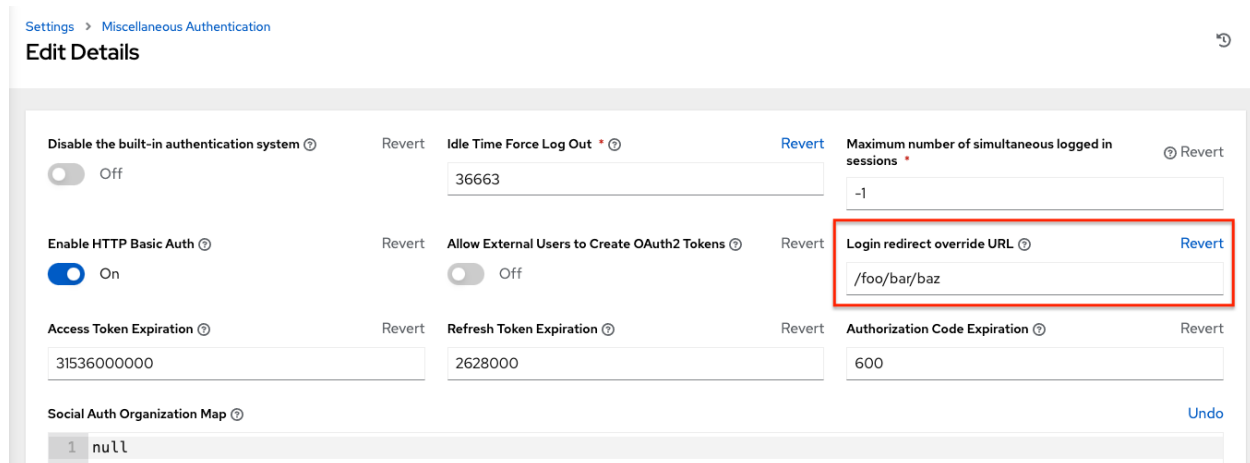
Alternatively, logout of automation controller and the login screen will now display the SAML logo to indicate it as a alternate method of logging into automation controller.



19.4.1 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, specify the redirect URL for non-logged-in users to somewhere other than the default controller login page by using the **Login redirect override URL** field in the Miscellaneous Authentication settings window of the **Settings** menu, accessible from the left navigation bar. This should be set to `/sso/login/saml/?idp=<name-of-your-idp>` for transparent SAML login, as shown in the example.



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 controller login page, including

SSO authentication buttons, and allows you to log in with any configured method.

19.5 TACACS+ 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 automation controller to use as a source for authentication.

Note: This feature is deprecated and will be removed in a future release.

1. Click **Settings** from the left navigation bar.
2. On the left side of the Settings window, click **TACACS+ settings** from the list of Authentication options.
3. Click **Edit** and 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**.

Settings > TACACS+ ↻

Edit Details

TACACS+ Server ⓘ	Revert	TACACS+ Port ⓘ	Revert	TACACS+ Secret ⓘ	Revert
<input type="text" value="tacacsplus.example.com"/>		<input type="text" value="49"/>		<input type="password" value="....."/>	
TACACS+ Auth Session Timeout ⓘ	Revert	TACACS+ Authentication Protocol ⓘ	Revert		
<input type="text" value="5"/>		<input type="text" value="ascii"/>			

4. Click **Save** when done.

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 the controller 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 controller 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 the controller:

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 controller 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 automation controller, however, you can install it from the `openldap-clients` package.

3. In the automation controller User Interface, click **Settings** from the left navigation and click to select **LDAP settings** from the list of Authentication options.

Note: You can configure multiple LDAP servers by specifying the server to configure (otherwise, leave the server at **Default**):

SETTINGS / AUTHENTICATION

AUTHENTICATION

AZURE AD GITHUB GOOGLE OAUTH2 **LDAP** RADIUS SAML TACACS+

LDAP SERVER Default

LDAP SERVER URI LDAP 1 (Optional) LDAP BIND DN LDAP BIND PASSWORD
ldaps://ldap.ex LDAP 2 (Optional) SHOW

LDAP USER DN TEMPLATE LDAP 3 (Optional) LDAP GROUP TYPE LDAP REQUIRE GROUP
uid=%(user)s, LDAP 4 (Optional) MemberDNGroupType CN=Tower Users,OU=Users,DC=example,DC=com

LDAP DENY GROUP LDAP 5 (Optional) LDAP START TLS
CN=Disabled Users,OU=Users,DC=example,DC=com

The equivalent API endpoints will show `AUTH_LDAP_*` repeated: `AUTH_LDAP_1_*`, `AUTH_LDAP_2_*`, ..., `AUTH_LDAP_5_*` to denote server designations.

- Click **Edit** and 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 ? REVERT

ldaps://ldap.domain.com:636

- Enter the Distinguished Name in the **LDAP Bind DN** text field to specify the user that the controller uses to connect (Bind) to the LDAP server. Below uses the example, `CN=josie,CN=users,DC=website,DC=com`:

LDAP BIND DN ? REVERT

CN=josie,CN=users,DC=website,DC=com

- 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 ?

SHOW

- 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)`.

8. 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

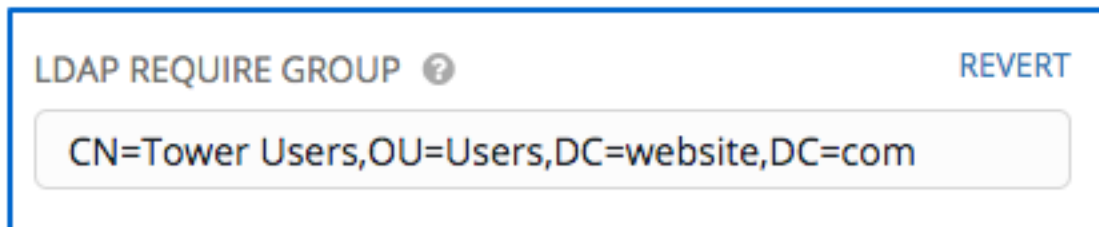
The LDAP Group Types that are supported by the controller leverage the underlying [django-auth-ldap](#) library.

Each **LDAP Group Type** can potentially take different parameters. The controller exposes `LDAP_GROUP_TYPE_PARAMS` to account for this. `LDAP_GROUP_TYPE_PARAMS` is a dictionary, which will be converted by the controller 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.

9. Enter the group distinguish name to allow users within that group to access the controller in the **LDAP Require Group** field, using the same format as the one shown in the text field. In this example, use: `CN=controller Users,OU=Users,DC=website,DC=com`



LDAP REQUIRE GROUP ? REVERT

CN=Tower Users,OU=Users,DC=website,DC=com

10. Enter the group distinguish name to prevent users within that group to access the controller 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.
11. The **LDAP Start TLS** is disabled by default. To enable TLS when the LDAP connection is not using SSL, click the toggle to **ON**.



12. 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:

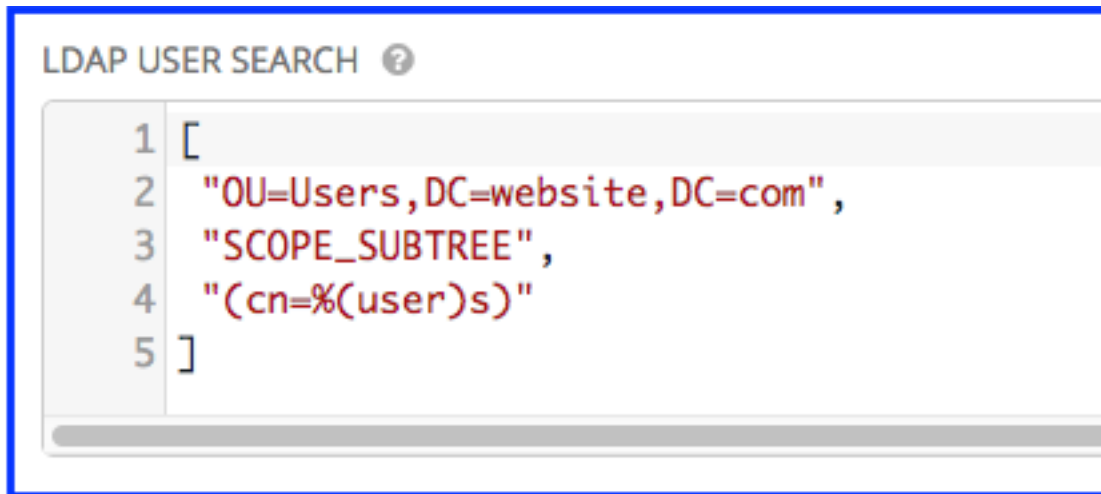
```
[
"OU=Users,DC=website,DC=com",
"SCOPE_SUBTREE",
"(cn=%(user)s)"
]
```

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.



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)

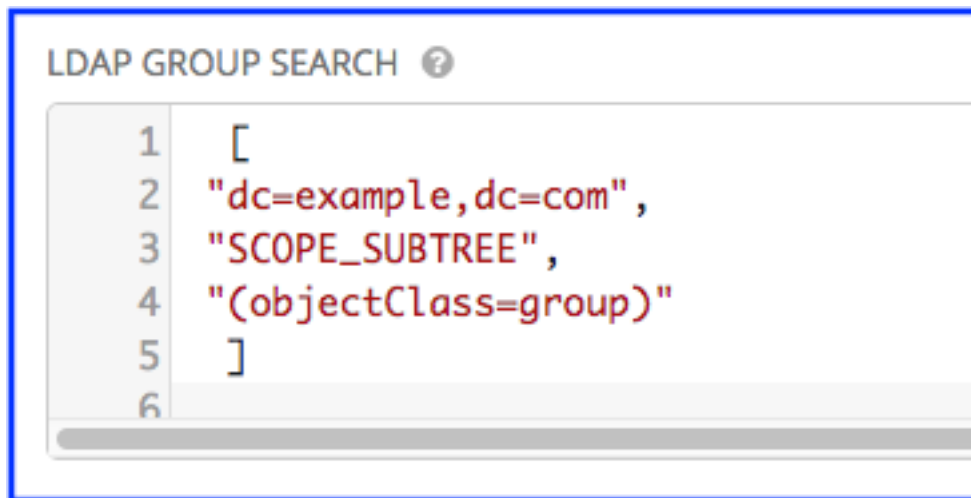
(continued from previous page)

```
[
  "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)"
]
]
```

13. 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.



14. 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.

```
LDAP USER ATTRIBUTE MAP ⊗ REVERT
1 {
2   "first_name": "givenName",
3   "last_name": "sn",
4   "email": "mail"
5 }
```

15. 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.

```
LDAP USER FLAGS BY GROUP ⊗ REVERT
1 {
2   "is_superuser": "cn=superusers,ou=groups,dc=website,dc=com",
3   "is_system_auditor": "cn=auditors,ou=groups,dc=website,dc=com"
4 }
```

16. For details on completing the mapping fields, see *LDAP Organization and Team Mapping*.
 17. Click **Save** when done.

With these values entered on this form, you can now make a successful authentication with LDAP.

Note: The controller does not actively sync users, but they are created during their initial login. To improve performance associated with LDAP authentication, see [LDAP authentication performance tips](#) in the *Automation Controller User Guide*.

20.1 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 automation controller version 2.4.3 and above. If you are running an earlier version of the controller, you should consider adding this parameter to your configuration file.

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

20.2 Enabling Logging for LDAP

To enable logging for LDAP, you must set the level to `DEBUG` in the Settings configuration window:

1. Click **Settings** from the left navigation pane and click to select **Logging settings** from the System list of options.
2. Click **Edit**.
3. Set the **Logging Aggregator Level Threshold** field to **Debug**.

4. Click **Save** to save your changes.

20.3 LDAP Organization and Team Mapping

Next, you will need to control which users are placed into which controller 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 organization.
- 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 a 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.


```

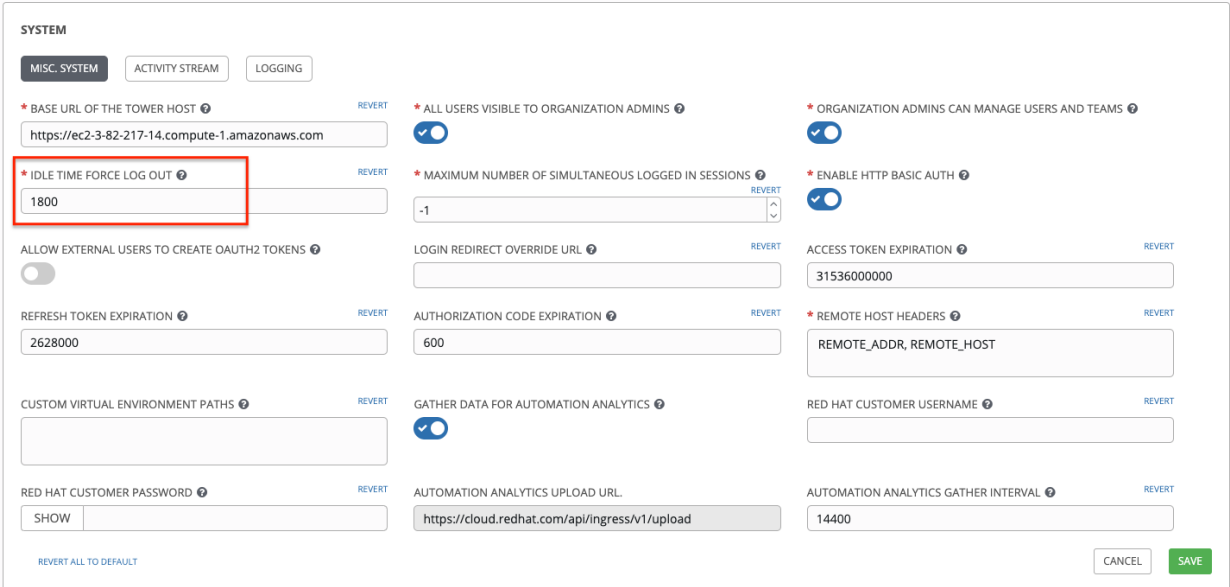
{
  "LDAP Engineering": {
    "organization": "LDAP Organization",
    "users": "cn=engineering,ou=groups,dc=example,dc=com",
    "remove": true
  },
  "LDAP IT": {
    "organization": "LDAP Organization",
    "users": "cn=it,ou=groups,dc=example,dc=com",
    "remove": true
  },
  "LDAP Sales": {
    "organization": "LDAP Organization",
    "users": "cn=sales,ou=groups,dc=example,dc=com",
    "remove": true
  }
}

```

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 controller 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.



SYSTEM

MISC. SYSTEM ACTIVITY STREAM LOGGING

* BASE URL OF THE TOWER HOST REVERT
https://ec2-3-82-217-14.compute-1.amazonaws.com

* IDLE TIME FORCE LOG OUT REVERT
1800

ALLOW EXTERNAL USERS TO CREATE OAUTH2 TOKENS REVERT

REFRESH TOKEN EXPIRATION REVERT
2628000

CUSTOM VIRTUAL ENVIRONMENT PATHS REVERT

RED HAT CUSTOMER PASSWORD REVERT
SHOW

* ALL USERS VISIBLE TO ORGANIZATION ADMINS REVERT

* ORGANIZATION ADMINS CAN MANAGE USERS AND TEAMS REVERT

* MAXIMUM NUMBER OF SIMULTANEOUS LOGGED IN SESSIONS REVERT
-1

LOGIN REDIRECT OVERRIDE URL REVERT

AUTHORIZATION CODE EXPIRATION REVERT
600

GATHER DATA FOR AUTOMATION ANALYTICS REVERT

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

* ORGANIZATION ADMINS CAN MANAGE USERS AND TEAMS REVERT

ENABLE HTTP BASIC AUTH REVERT

ACCESS TOKEN EXPIRATION REVERT
3153600000

* REMOTE HOST HEADERS REVERT
REMOTE_ADDR, REMOTE_HOST

RED HAT CUSTOMER USERNAME REVERT

AUTOMATION ANALYTICS GATHER INTERVAL REVERT
14400

REVERT ALL TO DEFAULT

4. Click **Save** to apply your changes.

Note: If you are accessing the controller 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.

USER AUTHENTICATION WITH KERBEROS

User authentication via Active Directory (AD), also referred to as authentication through Kerberos, is supported through the automation controller.

To get started, first set up the Kerberos packages in the controller 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 the controller, 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 the controller is easy. Generate the Kerberos ticket as the `awx` user and the controller 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.

22.1 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.

22.2 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 `krb5` `kinit`-compatible binary.

WORKING WITH SESSION LIMITS

Setting a session limit allows administrators to limit the number of simultaneous sessions per user or per IP address.

In automation controller, 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.

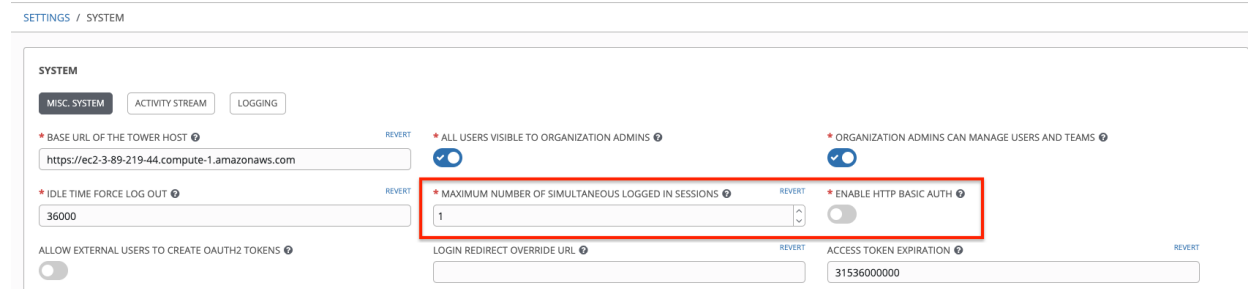
Session limits may be important, depending on your particular setup. For example, perhaps you only want a single user on your system with a single login per device (where the user could log in on his work laptop, phone, or home computer). In such a case, you would want to create a session limit equal 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 the **Miscellaneous System settings** of the Settings menu 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 controller UI, toggle the **Enable HTTP Basic Auth** to off.



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 automation controller instance.

BACKING UP AND RESTORING

The ability to backup and restore your system(s) is integrated into the platform 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 platform installation version. For example, if the current platform version you are on is 2.0.x, use only the latest 2.0 installer.

Also, backup and restore will *only* work on PostgreSQL versions supported by your current platform version. For more information, see [System Requirements](#).

The platform setup playbook is invoked as `setup.sh` from the path where you unpacked the platform 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 the platform 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>
```


24.1 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 the controller.

When restoring your system, the installer 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 controller host(s) are properly set up with SSH keys or user/pass variables in the hosts file, and that the user has sudo access.

24.2 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 platform 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 platform installation version. For example, if the current platform version you are on is `2.0.x`, use only the latest `2.0` 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.

24.3 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)
- Controller configuration (retrieved from `/etc/tower`)
- Controller secret key
- Manual projects

24.3.1 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 controller resources that were associated to instance groups will likely need to be reassigned to instance groups present on the new controller cluster.

USING CUSTOM LOGOS IN AUTOMATION CONTROLLER


automation controller 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  REVERT

Detailed

CUSTOM LOGO  REVERT

BROWSE Choose file

CUSTOM LOGIN INFO  REVERT

REVERT ALL TO DEFAULT CANCEL SAVE

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  REVERT

Off

CUSTOM LOGO  REVERT

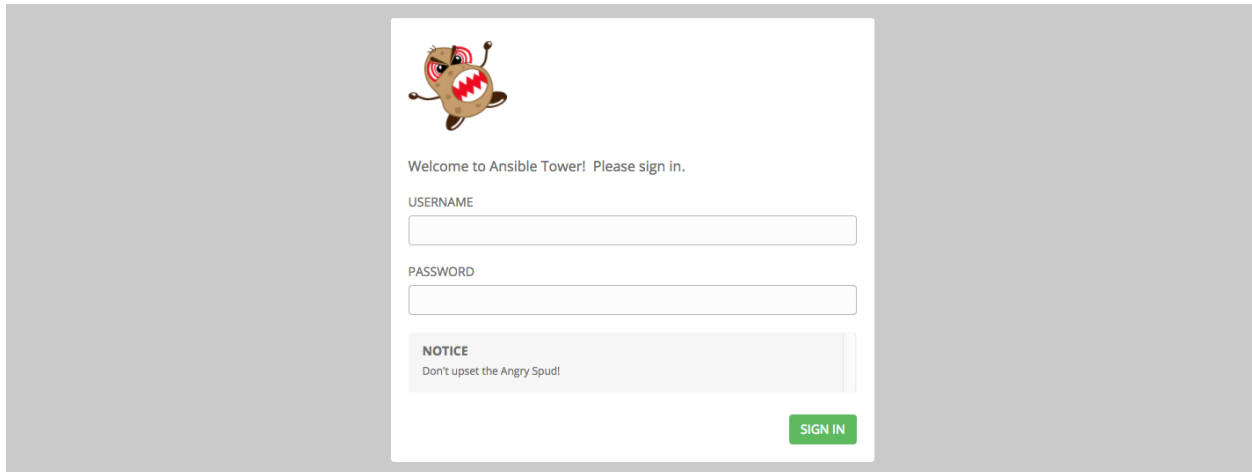
REMOVE angry-spud.png

CUSTOM LOGIN INFO  REVERT

Don't upset the Angry Spud!

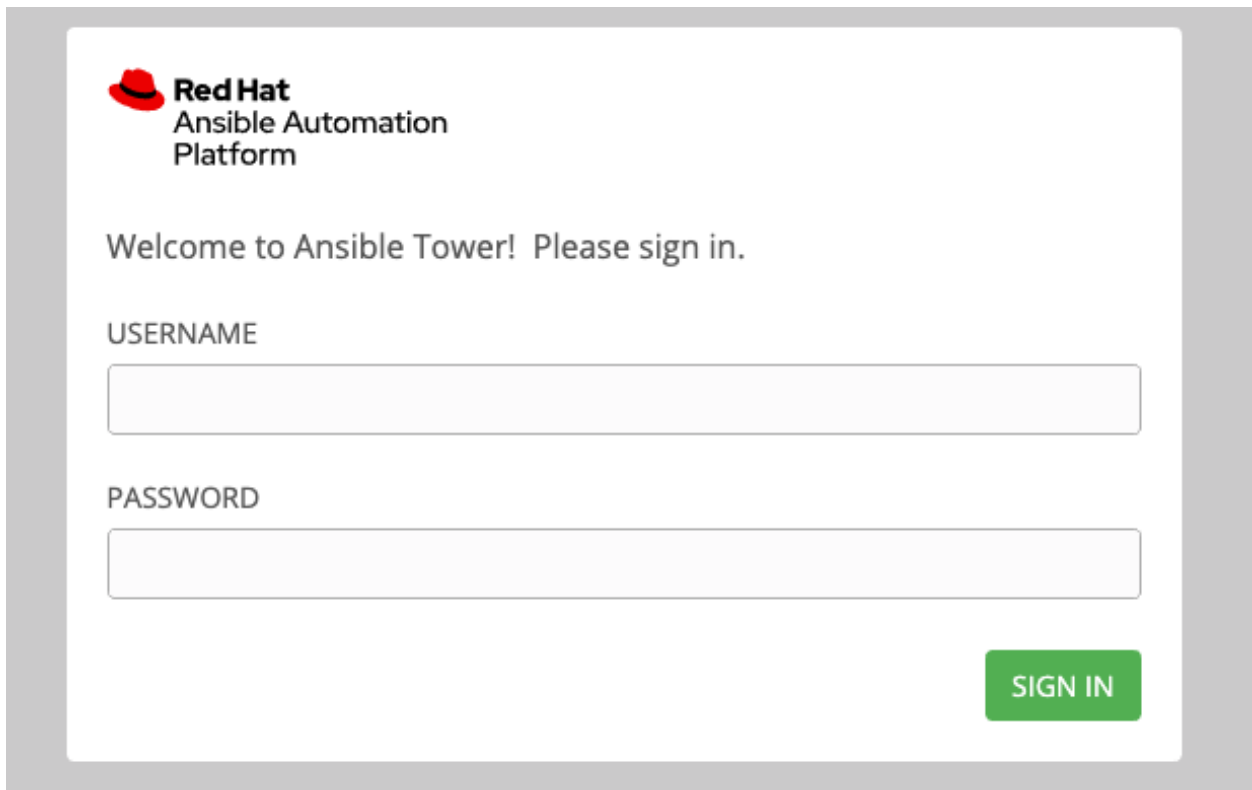
REVERT ALL TO DEFAULT CANCEL SAVE

The Tower login dialog would look like this:



The screenshot shows a login page with a white background centered on a gray background. At the top left is a cartoon mascot of a brown spud with a red face and sharp teeth. Below it is the text "Welcome to Ansible Tower! Please sign in." followed by two input fields labeled "USERNAME" and "PASSWORD". A gray notice box contains the text "NOTICE Don't upset the Angry Spud!". A green "SIGN IN" button is at the bottom right.

Selecting `Revert` will result in the appearance of the standard automation controller logo.



The screenshot shows a login page with a white background centered on a gray background. At the top left is the Red Hat logo (a red hat) followed by the text "Red Hat Ansible Automation Platform". Below it is the text "Welcome to Ansible Tower! Please sign in." followed by two input fields labeled "USERNAME" and "PASSWORD". A green "SIGN IN" button is at the bottom right.

USABILITY ANALYTICS AND DATA COLLECTION

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

Only users installing a trial of Red Hat Ansible Automation Platform or a fresh installation of automation controller 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 in the **User Interface settings**, by clicking **Settings** from the left navigation bar.

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

The screenshot shows the 'USER INTERFACE' settings page. It features three main configuration areas: 'USER ANALYTICS TRACKING STATE' with a dropdown menu showing 'Detailed', 'Off', 'Anonymous', and 'Detailed' (selected); 'CUSTOM LOGO' with a 'BROWSE' button and a 'Choose file' input field; and 'CUSTOM LOGIN INFO' with a large empty text area. At the bottom, there is a 'REVERT ALL TO DEFAULT' link on the left and 'CANCEL' and 'SAVE' buttons on the right.

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>.

26.1 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 automation controller **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 **Miscellaneous System settings** of the System configuration list of options located in the **Settings** menu.

SETTINGS / SYSTEM

SYSTEM

MISC: SYSTEM ACTIVITY STREAM LOGGING

* BASE URL OF THE TOWER HOST REVERT

* ALL USERS VISIBLE TO ORGANIZATION ADMINS REVERT

* ORGANIZATION ADMINS CAN MANAGE USERS AND TEAMS REVERT

* IDLE TIME FORCE LOG OUT REVERT

* MAXIMUM NUMBER OF SIMULTANEOUS LOGGED IN SESSIONS REVERT

* ENABLE HTTP BASIC AUTH REVERT

ALLOW EXTERNAL USERS TO CREATE OAUTH2 TOKENS REVERT

LOGIN REDIRECT OVERRIDE URL REVERT

ACCESS TOKEN EXPIRATION REVERT

REFRESH TOKEN EXPIRATION REVERT

AUTHORIZATION CODE EXPIRATION REVERT

* REMOTE HOST HEADERS REVERT

CUSTOM VIRTUAL ENVIRONMENT PATHS REVERT

GATHER DATA FOR AUTOMATION ANALYTICS REVERT

RED HAT CUSTOMER USERNAME REVERT

RED HAT CUSTOMER PASSWORD REVERT

AUTOMATION ANALYTICS UPLOAD URL REVERT

AUTOMATION ANALYTICS GATHER INTERVAL REVERT

REVERT ALL TO DEFAULT

CANCEL SAVE

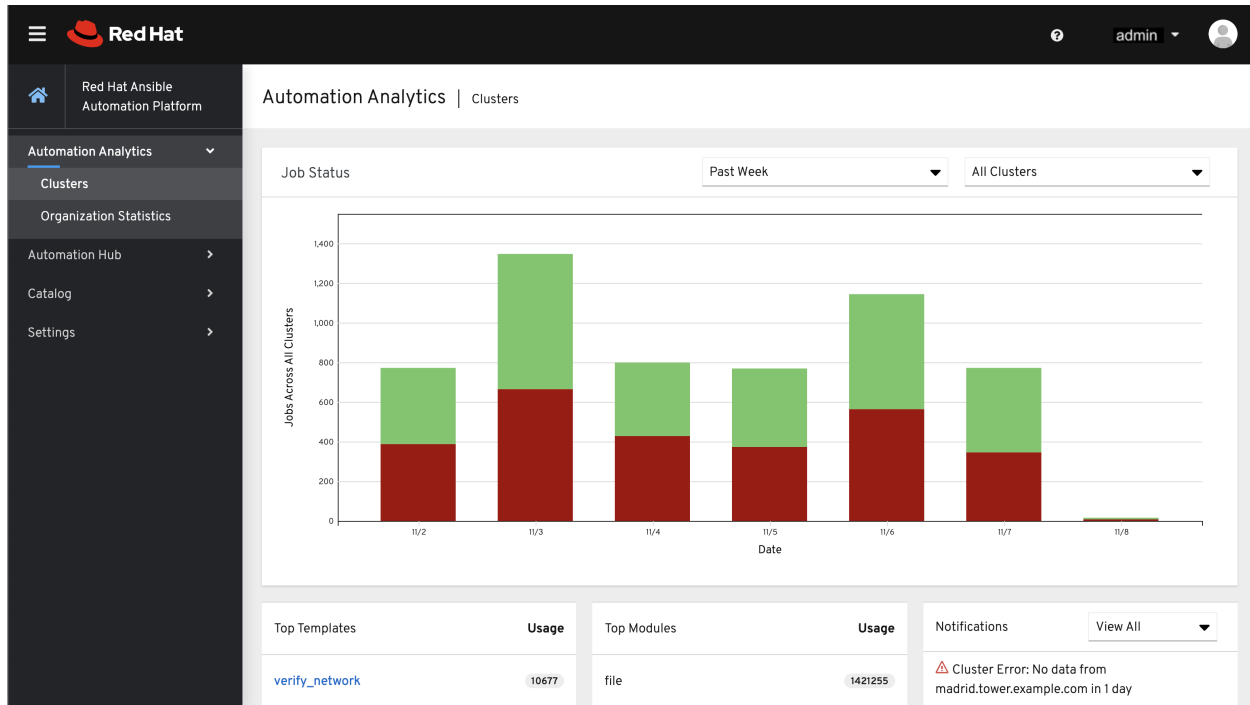
Note that the **Automation Analytics Upload URL** field is pre-populated with the location to which the collection of insights data will be uploaded.

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 **Miscellaneous System settings** 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:

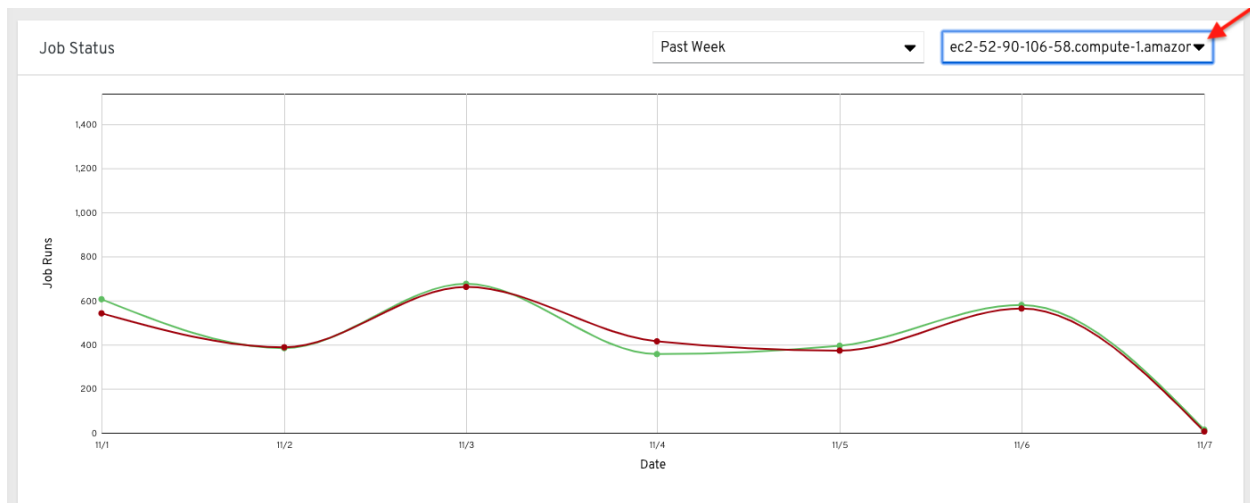
- `api/v2/settings/all`
- `api/v2/settings/system`

The Automation Analytics generated from this data collection will be found on the [Red Hat Cloud Services](#) portal.



The **Clusters** data is the default view. This graph represents the number of job runs across all controller 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.



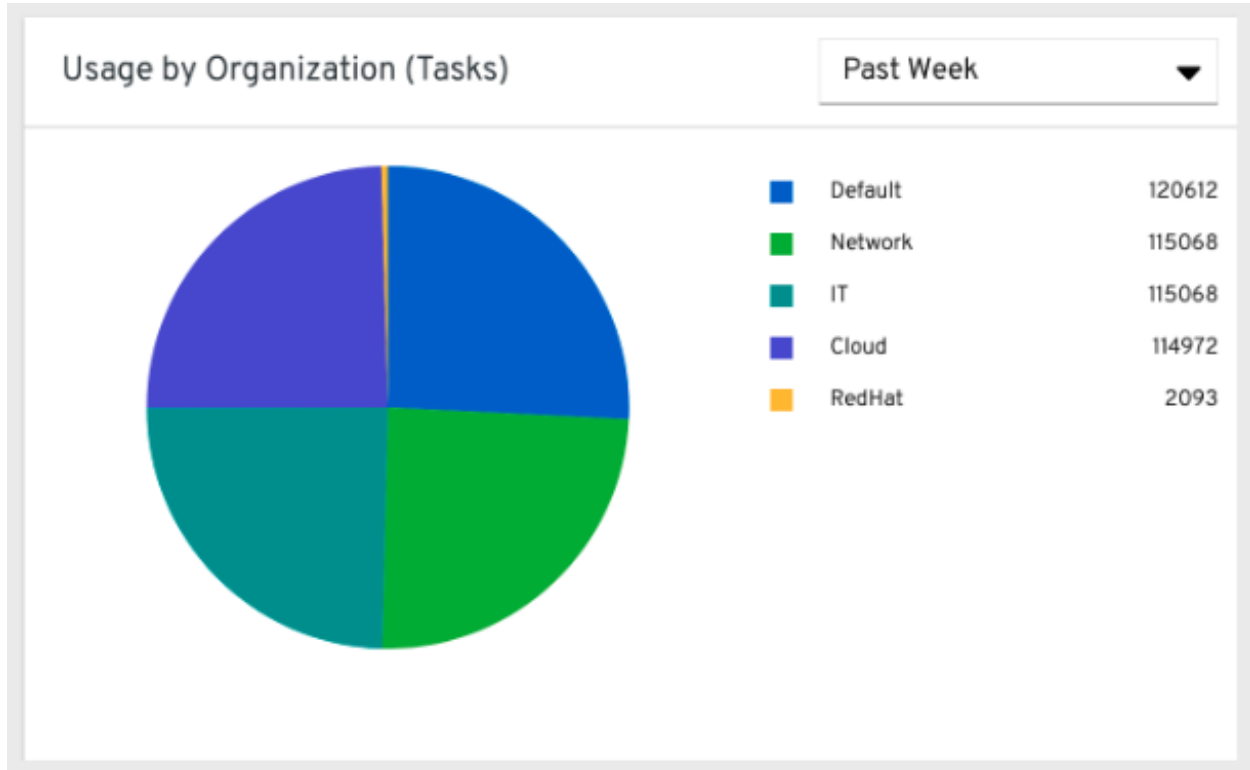
This multi-line chart represents the number of job runs for a single controller cluster for a specified period of time. The example here shows a span of a week, organized by the number of successfully running jobs (in green) and jobs that failed (in red). You can specify the number of successful and 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:

- *Usage by organization*
- *Job runs by organization*
- *Organization status*

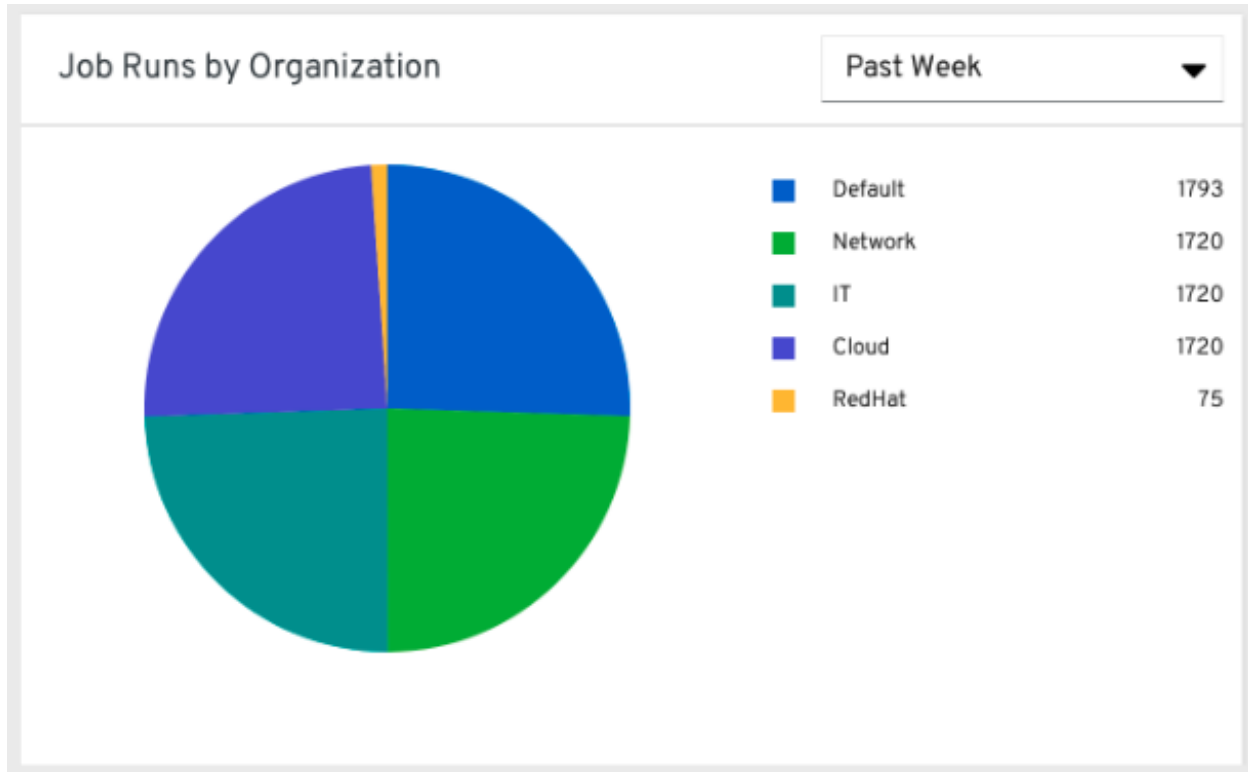
26.1.1 Usage by organization

This pie chart represents the number of tasks ran inside all jobs by a particular organization.



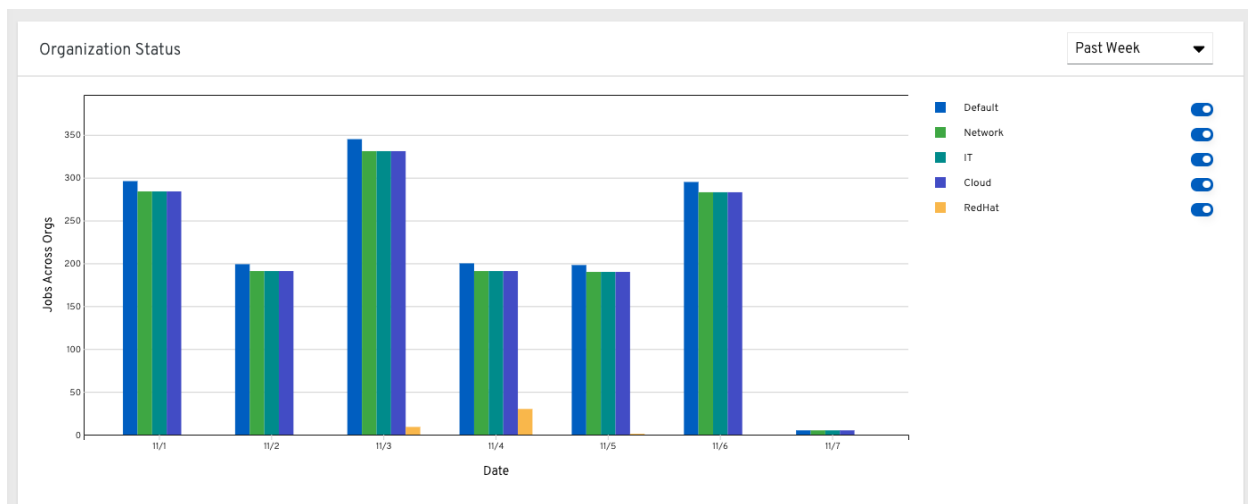
26.1.2 Job runs by organization

This pie chart represents the controller usage across *all* controller clusters by organization, which is calculated by the number of jobs run by that organization.



26.1.3 Organization status

This bar chart represents the controller 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.



26.2 Details of data collection

Automation Analytics collects certain classes of data from automation controller:

- Basic configuration, like which features are enabled, and what operating system is being used
- Topology and status of the controller 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 the controller 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.

- *manifest.json*
- *config.json*
- *instance_info.json*
- *counts.json*
- *org_counts.json*
- *cred_type_counts.json*
- *inventory_counts.json*
- *projects_by_scm_type.json*
- *query_info.json*
- *job_counts.json*
- *job_instance_counts.json*
- *unified_job_template_table.csv*
- *unified_jobs_table.csv*
- *workflow_job_template_node_table.csv*
- *workflow_job_node_table.csv*

- *events_table.csv*

26.2.1 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"
}
```

26.2.2 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,
"controller_url_base": "https://ansible.rhdemo.io",
"controller_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 *Logging and Aggregation* for details

logging_aggregators What logging categories are sent to external logging. See *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

controller_url_base The base URL for the cluster used by clients (shown in Automation Analytics)

controller_version Version of the software on the cluster

26.2.3 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",
    "version": "3.6.3"
  }
}

```

(continues on next page)

(continued from previous page)

```

}
"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

26.2.4 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,
  "user": 28,

```

(continues on next page)

(continued from previous page)

```

"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.

26.2.5 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
  }
}

```

26.2.6 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_controller": true,
    "name": "Machine"
  },
  "2": {
    "credential_count": 2,
    "managed_by_controller": true,
    "name": "Source Control"
  },
  "3": {
    "credential_count": 3,
    "managed_by_controller": true,
    "name": "Vault"
  },
  "4": {
    "credential_count": 0,
    "managed_by_controller": true,

```

(continues on next page)

(continued from previous page)

```

    "name": "Network"
  },
  "5": {
    "credential_count": 6,
    "managed_by_controller": true,
    "name": "Amazon Web Services"
  },
  "6": {
    "credential_count": 0,
    "managed_by_controller": true,
    "name": "OpenStack"
  },
  ...

```

26.2.7 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": {
    "num_hosts": 5,

```

(continues on next page)

(continued from previous page)

```

    "kind": "smart",
    "name": "Filtered AWS inventory",
    "source_list": [],
    "sources": 0
  }
}

```

26.2.8 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
}

```

26.2.9 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”.

26.2.10 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
}

```

26.2.11 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.

26.2.12 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

26.2.13 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', 'workflow', 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

26.2.14 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 workflow 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

26.2.15 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_template_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 succeeded

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

26.2.16 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

TROUBLESHOOTING THE CONTROLLER

27.1 Error logs

The controller 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 controller 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/>.

27.2 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.

27.3 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.)

27.4 Unable to login to the controller via HTTP

Access to the controller is intentionally restricted through a secure protocol (HTTPS). In cases where your configuration is set up to run a controller 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 controller instance:

```
SESSION_COOKIE_SECURE = False
CSRF_COOKIE_SECURE = False
```

Changing these settings to `False` will allow the controller 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:

```
automation-controller-service restart
```

27.5 WebSockets port for live events not working

automation controller uses port 80/443 on the controller 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.

27.6 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 `yamllint` 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.

27.7 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 the controller and should be set to `hosts: all`.

27.8 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/
```

27.9 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 `automation-controller-service restart` on the controller server.

If you continue to have problems, run `sosreport` as root on the controller server, then file a [support request](#) with the result.

27.10 Cancel a controller job

When issuing a `cancel` request on a currently running controller job, the controller 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.

27.11 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.

Automation controller uses container technology to isolate jobs from each other. By default, only the current project is exposed to the container running a job template.

You may find that you need to customize your playbook runs to expose additional directories. To fine tune your usage of job isolation, there are certain variables that can be set.

By default, automation controller 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 Jobs settings screen, or in the REST API at `/api/v2/settings/jobs`:

```
AWX_ISOLATION_BASE_PATH = "/opt/tmp"
```

If there are any additional directories that should specifically be exposed from the host to the container that playbooks run in, you can specify those in the **Paths to Expose to Isolated Jobs** field of the Jobs setting screen, or in the REST API at `/api/v2/settings/jobs`:

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

Note: The primary file you may want to add to `AWX_ISOLATION_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 REVERT

- 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 REVERT

* MAXIMUM SCHEDULED JOBS REVERT

PATHS TO EXPOSE TO ISOLATED JOBS REVERT

ANSIBLE CALLBACK PLUGINS REVERT

PATHS TO HIDE FROM ISOLATED JOBS REVERT

* ENABLE JOB ISOLATION REVERT

DEFAULT JOB TIMEOUT REVERT

DEFAULT INVENTORY UPDATE TIMEOUT REVERT

DEFAULT PROJECT UPDATE TIMEOUT REVERT

PER-HOST ANSIBLE FACT CACHE TIMEOUT REVERT

MAXIMUM NUMBER OF FORKS PER JOB REVERT

27.12 Private EC2 VPC Instances in the controller Inventory

By default, the controller 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 controller 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: The controller must be running inside the VPC with access to those instances if you want to configure them.

27.13 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 the controller, 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.

CONTROLLER TIPS AND TRICKS

- *Using the Controller CLI Tool*
- *Changing the Controller Admin Password*
- *Creating a controller Admin from the commandline*
- *Setting up a jump host to use with the controller*
- *View Ansible outputs for JSON commands when using the controller*
- *Locate and configure the Ansible configuration file*
- *View a listing of all ansible_ variables*
- *The ALLOW_JINJA_IN_EXTRA_VARS variable*
- *Using execution environments*
- *Configuring the controllerhost hostname for notifications*
- *Launching Jobs with curl*
- *Dynamic Inventory and private IP addresses*
- *Filtering instances returned by the dynamic inventory sources in the controller*
- *Using an unreleased module from Ansible source with the controller*
- *Using callback plugins with the controller*
- *Connecting to Windows with winrm*
- *Importing existing inventory files and host/group vars into the controller*

28.1 Using the Controller CLI Tool

Automation controller has a full-featured command line interface. Refer to [AWX Command Line Interface](#) documentation for configuration and usage instructions.

28.2 Changing the Controller 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 the controller. 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 controller 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.

28.3 Creating a controller 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 controller server and enter in the admin information as prompted:

```
awx-manage createsuperuser
```

28.4 Setting up a jump host to use with the controller

Credentials supplied by the controller 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
```

You can also add a jump host to your controller 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>"'
```

28.5 View Ansible outputs for JSON commands when using the controller

When working with automation controller, you can use the API to obtain the Ansible outputs for commands in JSON format.

To view the Ansible outputs, browse to:

```
https://<controller server name>/api/v2/jobs/<job_id>/job_events/
```

28.6 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. Automation controller 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.

The controller overrides some `ansible.cfg` options. For example, the controller stores the SSH ControlMaster sockets, the SSH agent socket, and any other per-job run items in a per-job temporary directory that is passed to the container used for job execution.

28.7 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

28.8 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).

28.9 Using execution environments

See [Execution Environments](#) in the *Automation Controller User Guide*.

28.10 Configuring the controllerhost hostname for notifications

In the *System Settings*, you can replace `https://controller.example.com` in the **Base URL of The Controller Host** field with your preferred hostname to change the notification hostname.

Settings > Miscellaneous System ⌵

Edit Details

Enable Activity Stream ⓘ <input checked="" type="checkbox"/> On	Revert	Enable Activity Stream for Inventory Sync ⓘ <input type="checkbox"/> Off	Revert	Global default execution environment ⓘ <input type="text" value=""/>	Revert
Base URL of the service * ⓘ <input type="text" value="https://towerhost"/>	Revert	All Users Visible to Organization Admins ⓘ <input checked="" type="checkbox"/> On	Revert	Organization Admins Can Manage Users and Teams ⓘ <input checked="" type="checkbox"/> On	Revert
Gather data for Automation Analytics ⓘ <input checked="" type="checkbox"/> On	Revert	Red Hat customer username ⓘ <input type="text" value=""/>	Southwest-05-22-22.pdf revert	Red Hat customer password ⓘ <input type="password" value=""/>	Revert
Red Hat or Satellite username ⓘ <input type="text" value="thavo@redhat.com"/>	Revert	Red Hat or Satellite password ⓘ <input type="password" value="ENCRYPTED"/>	Revert	Automation Analytics Gather Interval * ⓘ <input type="text" value="14400"/>	Revert
Last gathered entries from the data collection service of Automation Analytics					Revert
<input type="text" value="1"/>					

Refreshing your controller license also changes the notification hostname. New installations of automation controller should not have to set the hostname for notifications.

28.11 Launching Jobs with curl

Launching jobs with the controller API is simple. Here are some easy to follow examples using the `curl` tool.

Assuming that your Job Template ID is '1', your controller 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.

28.12 Dynamic Inventory and private IP addresses

By default, the controller 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 controller 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: The controller must be running inside the VPC with access to those instances in order to usefully configure them.

28.13 Filtering instances returned by the dynamic inventory sources in the controller

By default, the dynamic inventory sources in the controller (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.

28.14 Using an unreleased module from Ansible source with the controller

If there is a feature that is available in the latest Ansible core branch that you would like to leverage with your controller system, making use of it in the controller 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.

28.15 Using callback plugins with the controller

Ansible has a flexible method of handling actions during playbook runs, called callback plugins. You can use these plugins with the controller 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: automation controller does not support the `stdout` callback plugin because Ansible only allows one, and it is already being used by automation controller 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 controller Project. Then, specify their paths (one path per line) in the **Ansible Callback Plugins** field of the Job settings, located towards the bottom of the screen:


The screenshot shows a configuration window for 'Ansible Callback Plugins'. At the top, there are two input fields with values '20' and '21', and a label '"win_user"'. Below this is a section titled 'Ansible Callback Plugins' with a 'Revert' button. It contains a single entry with the value '1' and a small icon. Below this is a section titled 'Paths to expose to isolated jobs' with a 'Revert' button and a single entry with the value '1'. Below that is a section titled 'Extra Environment Variables' with a 'Revert' button and a single entry with a small icon. At the bottom of the window are three buttons: 'Save', 'Revert all to default', and 'Cancel'.

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](#).

28.16 Connecting to Windows with winrm

By default controller 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.

28.17 Importing existing inventory files and host/group vars into the controller

To import an existing static inventory and the accompanying host and group vars into the controller, your inventory should be in a structure that looks similar to the following:

```
inventory/
|-- group_vars
|   `-- mygroup
|-- host_vars
|   `-- myhost
`-- hosts
```

To import these hosts and vars, run the `awx-manage` command:

```
awx-manage inventory_import --source=inventory/ \
  --inventory-name="My Controller 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 Controller Inventory"
```

In case of conflicts or to overwrite an inventory named “My Controller Inventory”, run:

```
awx-manage inventory_import --source=inventory/ \
  --inventory-name="My Controller 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.

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, Automation controller, controller CLI, and Ansible Galaxy are all, as Doge would say, “much approved.”¹

¹ <http://knowyourmeme.com/memes/doge>



Josie Tested - Doge Approved.

- genindex

COPYRIGHT © RED HAT, INC.

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

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

Third Party Rights

Ubuntu and Canonical are registered trademarks of Canonical Ltd.

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

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

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

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

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

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

Safari® is a registered trademark of Apple, Inc.

Firefox® is a registered trademark of the Mozilla Foundation.

All other trademarks are the property of their respective owners.

A

- Active Directory (*AD*)
 - Kerberos, 111
- activity stream cleanup management
 - job, 15
- add
 - execution environment, 144
- add execution environment
 - jobs, 144
- admin creation
 - commandline, 142
 - tips, 142
- admin password
 - changing password, 142
- admin password change
 - tips, 142
- admin utility script, 5
- Analytics
 - Insights, 120
- analytics collection, 64, 120
- Ansible configuration file, 143
- Ansible modules, unreleased
 - tips, 146
- Ansible output for JSON commands, 143
- ansible.cfg, 143
 - tips, 143
- ansible_variables, viewing all
 - tips, 143
- ansible-controller script replacement, 5
- API
 - instance group, 27
- assignment
 - credentials, 10
- attaching
 - subscription, 3
- AUTH_BASIC_ENABLED
 - session limits, 114
- authentication, 60, 71, 81, 91
 - Azure AD, 91
 - configuration, 60
 - GitHub Enterprise, 84

- GitHub Enterprise Org, 85
- GitHub Enterprise Team, 86
- GitHub OAuth2, 81
- GitHub Org, 82
- GitHub Team, 83
- Google OAuth2, 87
- LDAP, 102, 108
 - LDAP mapping, 108
 - LDAP team mapping, 108
 - organization mapping, 89, 108
- RADIUS Authentication Settings, 92
- SAML, 100
 - SAML Service Provider, 93
- TACACS+ Authentication Settings, 101
- team mapping, 89, 108
- authentication expiring, 110
- authentication timeout
 - changing the default, 110
 - troubleshooting, 110
- authentication token, 110
- Automation
 - Insights, 120
- automationcontroller groups
 - policies, 26
- awx-manage, 56
 - analytics gathering, 59
 - change password, 142
 - cluster management, 57
 - data collection, 59
 - inventory import, 56
 - session management, 57
 - super user creation, 142
 - token management, 57
- awx-manage, data cleanup, 57
- Azure AD
 - authentication, 91

B

- backups, 115
 - considerations, 116
 - playbooks, 116
- best practices, 141

C

- callback plugins
 - tips, 146
- capacity
 - container groups, 36
- Centos
 - clustering, 20
- change password
 - awx-manage, 142
- changing password
 - admin password, 142
- changing the default
 - authentication timeout, 110
- cleaning old data, 14
- cleanup activity stream
 - management jobs, 15
- cleanup expired OAuth2 tokens
 - management jobs, 18
- cleanup expired sessions
 - management jobs, 18
- cleanup job history
 - management jobs, 18
- cluster
 - deprovisioning, 24
- clustering
 - backup, 117
 - Centos, 20
 - instance group policies, 27
 - operating systems, 20
 - pinning, 28
 - PostgreSQL, 20
 - redundancy, 20
 - restore, 117
 - RHEL, 20
 - setup considerations, 20
 - Ubuntu, 20
- command line interface
 - controller CLI, 141
 - tips, 141
- commandline
 - admin creation, 142
- components
 - licenses, 4
- configuration
 - authentication, 60
 - custom login message, 63
 - custom logo, 63
 - data collection, 63
 - jobs, 61
 - system, 62
 - UI, 63
- configuration file configuration
 - tips, 143
- configuration file location
 - tips, 143
- configure
 - instance group, 27
- configure controller, 60
- consume
 - subscription, 3
- container
 - groups, 25
- container groups, 31
 - capacity, 36
 - limits, 36
- containers
 - instance groups, 31
- controller admin utility script, 5
- controller CLI
 - command line interface, 141
 - tips, 141
- credentials, 10
 - assignment, 10
 - multi, 10
- curl
 - tips, 144
- custom
 - login message, 63
 - logo, 63, 118
- custom inventory scripts, 6
- custom login message
 - configuration, 63
- custom logo, 118
 - configuration, 63

D

- data collection, 64, 120
 - configuration, 63
- DEB files
 - licenses, 4
- deprovisioning
 - cluster, 24
 - groups, 30
- dynamic inventory and instance
 - filtering
 - tips, 145
- dynamic inventory and private IPs
 - tips, 145

E

- EC2
 - VPC instances, 139
- EC2 VPC instances
 - tips, 145
 - troubleshooting, 139
- Elastic stack
 - logging, 39
- ELK stack

- logging, 39
- enterprise authentication, 60, 91
- error logs
 - troubleshooting, 136
- evaluation, 2
- execution environment, 144
 - add, 144
- expired OAuth2 tokens cleanup
 - management job, 18
- expired sessions cleanup management
 - job, 18
- external database
 - installation failure, 138

F

- features, 1
- filtering instances
 - tips, 145
- functionality
 - isolation, 70

G

- general help
 - troubleshooting, 136
- GitHub Enterprise
 - authentication, 84
- GitHub Enterprise Org
 - authentication, 85
- GitHub Enterprise Team
 - authentication, 86
- GitHub OAuth2
 - authentication, 81
- GitHub Org
 - authentication, 82
- GitHub Team
 - authentication, 83
- Google OAuth2
 - authentication, 87
- groups
 - container, 25
 - deprovisioning, 30
 - instance, 25

H

- handling
 - secret key, 49
- help, 136, 141
- host connections
 - troubleshooting, 136, 137
- host/group vars import
 - tips, 147
- hostname configuration
 - notifications, 144
- hosts list

- troubleshooting, 140
- hosts lists (*empty*), 140

I

- import
 - license, 66
- importing host/group vars
 - importing inventory, 147
- importing inventory
 - importing host/group vars, 147
- init script replacement, 5
- Insights
 - Analytics, 120
 - Automation, 120
- installation bundle
 - licenses, 4
- installation failure
 - external database, 138
- installation wizard
 - playbook backup/restore arguments, 115
- instance
 - groups, 25
- instance filtering
 - tips, 145
- instance group
 - API, 27
 - configure, 27
- instance group policies
 - clustering, 27
- instance groups
 - containers, 31
 - pinning, 28
 - policies, 27
 - redundancy, 20
- inventory file importing, 7
- inventory import
 - tips, 147
- inventory scripts
 - custom, 6, 7
- isolation
 - functionality, 70
 - troubleshooting, 70
 - variables, 70

J

- job cancellation
 - troubleshooting, 138
- job does not run
 - troubleshooting, 137
- job history cleanup management job, 18
- Job Template drop-down list
 - playbooks are not viewable, 138
- jobs, 61

- add execution environment, 144
- configuration, 61
- JSON commands, Ansible output, 143
- jump host
 - ProxyCommand, 142
 - tips, 142

K

- Kerberos
 - Active Directory (AD), 111
 - user authentication, 111
- keys, 49

L

- LDAP, 102, 108
 - authentication, 102, 108
 - referrals, 107
- LDAP mapping, 108
 - authentication, 108
- LDAP referrals
 - troubleshooting, 107
- LDAP team mapping
 - authentication, 108
- license, 1, 2
 - import, 66
 - nodes, 3
 - trial, 2
 - types, 2
 - UI, 66
- license features, 1
- licenses
 - components, 4
 - DEB files, 4
 - installation bundle, 4
 - RPM files, 4
- limits
 - container groups, 36
- live events
 - port changes, 137
 - troubleshooting, 137
- log, 110
- logfiles, 37
- logging, 39
 - Elastic stack, 39
 - ELK stack, 39
 - loggly, 39
 - logstash, 39
 - rsyslog, 39
 - schema, 39
 - splunk, 39
 - sumologic, 39
- loggly
 - logging, 39
- login message

- custom, 63
- login timeout, 110
- logo
 - custom, 63, 118
- logstash
 - logging, 39

M

- management jobs, 14
 - cleanup activity stream, 15
 - cleanup expired OAuth2 tokens, 18
 - cleanup expired sessions, 18
 - cleanup job history, 18
- metrics
 - prometheus, 47
- modules, using unreleased
 - tips, 146
- multi
 - credentials, 10

N

- notifications
 - hostname configuration, 144

O

- organization mapping, 89, 108
 - authentication, 89, 108

P

- pending playbook
 - troubleshooting, 138
- Pendo, 64, 120
- pinning
 - clustering, 28
 - instance groups, 28
- playbook setup
 - backup/restore arguments, 115
- playbooks are not viewable
 - Job Template drop-down list, 138
- playbooks not appearing
 - troubleshooting, 138
- plugins, callback
 - tips, 146
- policies
 - automationcontroller groups, 26
 - instance groups, 27
- port changes
 - live events, 137
- postface, 149
- PostgreSQL
 - clustering, 20
- private IPs with dynamic inventory
 - tips, 145
- prometheus

- metrics, 47
- ProxyCommand
 - jump host, 142
 - tips, 142

R

- RADIUS Authentication Settings
 - authentication, 92
- rebranding, 118
- redundancy
 - clustering, 20
 - instance groups, 20
- referrals
 - LDAP, 107
- regenerate
 - secret key, 49
- removing old data, 14
- restart controller, 5
- restorations, 115
 - considerations, 116
 - playbooks, 116
- restore
 - clustering, 117
- RHEL
 - clustering, 20
- RPM files
 - licenses, 4
- rsyslog
 - logging, 39

S

- SAML
 - authentication, 100
 - transparent, 100
- SAML Service Provider
 - authentication, 93
- schema
 - logging, 39
- scripts, admin utility, 5
- secret key
 - handling, 49
 - regenerate, 49
- session
 - timeout, 110
- session limits, 114
 - AUTH_BASIC_ENABLED, 114
 - SESSIONS_PER_USER, 114
- session.py, 114
- SESSIONS_PER_USER
 - session limits, 114
- social authentication, 60, 81
- sosreport
 - troubleshooting, 136
- splunk

- logging, 39
- start controller, 5
- stop controller, 5
- subscription
 - attaching, 3
 - consume, 3
- sumologic
 - logging, 39
- super user creation
 - awx-manage, 142
- support, 1, 2
- system
 - configuration, 62

T

- TACACS+ Authentication Settings
 - authentication, 101
- team mapping, 89, 108
 - authentication, 89, 108
- timeout
 - session, 110
- timeout login, 110
- tips, 141
 - admin creation, 142
 - admin password change, 142
 - Ansible modules, unreleased, 146
 - ansible.cfg, 143
 - ansible_variables, viewing all, 143
 - callback plugins, 146
 - command line interface, 141
 - configuration file configuration, 143
 - configuration file location, 143
 - controller CLI, 141
 - curl, 144
 - dynamic inventory and instance filtering, 145
 - dynamic inventory and private IPs, 145
 - EC2 VPC instances, 145
 - filtering instances, 145
 - host/group vars import, 147
 - instance filtering, 145
 - inventory import, 147
 - jump host, 142
 - modules, using unreleased, 146
 - plugins, callback, 146
 - private IPs with dynamic inventory, 145
 - ProxyCommand, 142
 - unreleased modules, 146
 - Windows connection, 147
 - winrm, 147
- token-based authentication, 71

- transparent
 - SAML, 100
- trial, 2
- troubleshooting, 136
 - authentication timeout, 110
 - EC2 VPC instances, 139
 - error logs, 136
 - general help, 136
 - host connections, 136, 137
 - hosts list, 140
 - isolation, 70
 - job cancellation, 138
 - job does not run, 137
 - LDAP referrals, 107
 - live events, 137
 - pending playbook, 138
 - playbooks not appearing, 138
 - sosreport, 136
 - websockets, 137

U

- Ubuntu
 - clustering, 20
- UI
 - configuration, 63
 - license, 66
- unreleased modules
 - tips, 146
- updates, 2
- usability data collection, 64, 120
- user authentication
 - Kerberos, 111
- user data tracking, 64, 120
- USER_ANALYTICS_TRACKING_STATE, 64, 120

V

- variables
 - isolation, 70
- VPC instances
 - EC2, 139

W

- websockets
 - troubleshooting, 137
- Windows connection
 - tips, 147
- winrm
 - tips, 147