Red Hat Ansible Automation Platform 2.5

Using automation execution

Use automation execution deploy, define, operate, scale and delegate automation

Last Updated: 2024-10-15

Red Hat Ansible Automation Platform 2.5 Using automation execution

Use automation execution deploy, define, operate, scale and delegate automation

Legal Notice

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This guide shows you how to use automation controller to define, operate, scale and delegate automation across your enterprise.

Table of Contents

PREFACE

Thank you for your interest in Red Hat Ansible Automation Platform automation controller. Automation controller helps teams manage complex multitiered deployments by adding control, knowledge, and delegation to Ansible-powered environments.

Using automation controller describes all of the functionality available in automation controller. It assumes moderate familiarity with Ansible, including concepts such as playbooks, variables, and tags. For more information about these and other Ansible concepts, see the Ansible documentation.

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

If you have a suggestion to improve this documentation, or find an error, you can contact technical support at https://access.redhat.com to open a request.

CHAPTER 1. AUTOMATION CONTROLLER OVERVIEW

With Ansible Automation Platform users across an organization can share, vet, and manage automation content by means of a simple, powerful, and agentless technical implementation. IT managers can provide guidelines on how automation is applied to individual teams. Automation developers can write tasks that use existing knowledge, without the operational overhead of conforming to complex tools and frameworks. It is a more secure and stable foundation for deploying end-to-end automation solutions, from hybrid cloud to the edge.

Ansible Automation Platform includes automation controller, which enables users to define, operate, scale, and delegate automation across their enterprise.

1.1. REAL-TIME PLAYBOOK OUTPUT AND EXPLORATION

With automation controller you can watch playbooks run in real time, seeing each host as they check in. You can go back and explore the results for specific tasks and hosts in great detail, search for specific plays or hosts and see just those results, or locate errors that need to be corrected.

1.2. "PUSH BUTTON" AUTOMATION

Use automation controller to access your favorite projects and re-trigger execution from the web interface. Automation controller asks for input variables, prompts for your credentials, starts and monitors jobs, and displays results and host history.

1.3. SIMPLIFIED ROLE-BASED ACCESS CONTROL AND AUDITING

With automation controller you can:

- Grant permissions to perform a specific task to different teams or explicit users through *role-based access control* (RBAC). Example tasks include viewing, creating, or modifying a file.
- Keep some projects private, while enabling some users to edit inventories, and others to run playbooks against certain systems, either in check (dry run) or live mode.
- Enable certain users to use credentials without exposing the credentials to them.

Automation controller records the history of operations and who made them, including objects edited and jobs launched.

If you want to give any user or team permissions to use a job template, you can assign permissions directly on the job template. Credentials are full objects in the automation controller RBAC system, and can be assigned to many users or teams for use.

Automation controller includes an *auditor* type. A system-level auditor can see all aspects of the systems automation, but does not have permission to run or change automation. An auditor is useful for a service account that scrapes automation information from the REST API.

Additional resources

- For more information about user roles, see Managing access with role based access control .

1.4. CLOUD AND AUTOSCALING FLEXIBILITY

Automation controller includes a powerful optional provisioning callback feature that enables nodes to request configuration on-demand. This is an ideal solution for a cloud auto-scaling scenario and includes the following features:

- It integrates with provisioning servers such as Cobbler and deals with managed systems with unpredictable uptimes.
- It requires no management software to be installed on remote nodes.
- The callback solution can be triggered by a call to **curl** or **wget**, and can be embedded in **init** scripts, kickstarts, or preseeds.
- You can control access so that only machines listed in the inventory can request configuration.

1.5. THE IDEAL RESTFUL API

The automation controller REST API is the ideal RESTful API for a systems management application, with all resources fully discoverable, paginated, searchable, and well modeled. A styled API browser enables API exploration from the API root at **http://<server name>/api/**, showing off every resource and relation. Everything that can be done in the user interface can be done in the API.

1.6. BACKUP AND RESTORE

Ansible Automation Platform can backup and restore your systems or systems, making it easy for you to backup and replicate your instance as required.

1.7. ANSIBLE GALAXY INTEGRATION

By including an Ansible Galaxy **requirements.yml** file in your project directory, automation controller automatically fetches the roles your playbook needs from Galaxy, GitHub, or your local source control. For more information, see Ansible Galaxy Support .

1.8. INVENTORY SUPPORT FOR OPENSTACK

Dynamic inventory support is available for OpenStack. With this you can target any of the virtual machines or images running in your OpenStack cloud.

For more information, see OpenStack credential type.

1.9. REMOTE COMMAND EXECUTION

Use remote command execution to perform a simple tasks, such as adding a single user, updating a single security vulnerability, or restarting a failing service. Any task that you can describe as a single Ansible play can be run on a host or group of hosts in your inventory. You can manage your systems quickly and easily. Because of an RBAC engine and detailed audit logging, you know which user has completed a specific task.

1.10. SYSTEM TRACKING

You can collect facts using the fact caching feature. For more information, see Fact Caching.

1.11. INTEGRATED NOTIFICATIONS

Keep track of the status of your automation.

You can configure the following notifications:

- stackable notifications for job templates, projects, or entire organizations
- different notifications for job start, job success, job failure, and job approval (for workflow nodes)

The following notification sources are supported:

- Email
- Grafana
- IRC
- Mattermost
- PagerDuty
- Rocket.Chat
- Slack
- Twilio
- Webhook (post to an arbitrary webhook, for integration into other tools)

You can also customize notification messages for each of the preceding notification types.

1.12. INTEGRATIONS

Automation controller supports the following integrations:

- Dynamic inventory sources for Red Hat Satellite 6.

For more information, see Red Hat Satellite 6 .

- Red Hat Insights integration, enabling Insights playbooks to be used as an Ansible Automation Platform project.

For more information, see Setting up Red Hat Insights for Red Hat Ansible Automation Platform Remediations.

- Automation hub acts as a content provider for automation controller, requiring both an automation controller deployment and an automation hub deployment running alongside each other.

1.13. CUSTOM VIRTUAL ENVIRONMENTS

With Custom Ansible environment support you can have different Ansible environments and specify custom paths for different teams and jobs.

1.14. AUTHENTICATION ENHANCEMENTS

Automation controller supports:

- LDAP
- SAML
- token-based authentication

With LDAP and SAML support you can integrate your enterprise account information in a more flexible manner.

Token-based authentication permits authentication of third-party tools and services with automation controller through integrated OAuth 2 token support.

1.15. CLUSTER MANAGEMENT

Run time management of cluster groups enables configurable scaling.

1.16. WORKFLOW ENHANCEMENTS

To model your complex provisioning, deployment, and orchestration workflows, you can use automation controller expanded workflows in several ways:

- **Inventory overrides for Workflows**You can override an inventory across a workflow at workflow definition time, or at launch time. Use automation controller to define your application deployment workflows, and then re-use them in many environments.
- **Convergence nodes for Workflows**When modeling complex processes, you must sometimes wait for many steps to finish before proceeding. Automation controller workflows can replicate this; workflow steps can wait for any number of earlier workflow steps to complete properly before proceeding.
- **Workflow Nesting** You can re-use individual workflows as components of a larger workflow. Examples include combining provisioning and application deployment workflows into a single workflow.
- **Workflow Pause and Approval**You can build workflows containing approval nodes that require user intervention. This makes it possible to pause workflows in between playbooks so that a user can give approval (or denial) for continuing on to the next step in the workflow.

For more information, see Workflows in automation controller .

1.17. JOB DISTRIBUTION

Take a fact gathering or configuration job running across thousands of machines and divide it into slices that can be distributed across your automation controller cluster. This increases reliability, offers faster job completion, and improved cluster use.

For example, you can change a parameter across 15,000 switches at scale, or gather information across your multi-thousand-node RHEL estate.

For more information, see Job Slicing.

1.18. SUPPORT FOR DEPLOYMENT IN A FIPS-ENABLED ENVIRONMENT

Automation controller deploys and runs in restricted modes such as FIPS.

1.19. LIMIT THE NUMBER OF HOSTS PER ORGANIZATION

Many large organizations have instances shared among many organizations. To ensure that one organization cannot use all the licensed hosts, this feature enables superusers to set a specified upper limit on how many licensed hosts can that you can allocate to each organization. The automation controller algorithm factors changes in the limit for an organization and the number of total hosts across all organizations. Inventory updates fail if an inventory synchronization brings an organization out of compliance with the policy. Additionally, superusers are able to over-allocate their licenses, with a warning.

1.20. INVENTORY PLUGINS

The following inventory plugins are used from upstream collections:

- **amazon.aws.aws_ec2**
- **community.vmware.vmware_vm_inventory**
- **azure.azcollection.azure_rm**
- **google.cloud.gcp_compute**
- **theforeman.foreman.foreman**
- **openstack.cloud.openstack**
- **ovirt.ovirt.ovirt**
- **awx.awx.tower**

1.21. SECRET MANAGEMENT SYSTEM

With a secret management system, external credentials are stored and supplied for use in automation controller so you need not provide them directly.

CHAPTER 2. LOGGING INTO AUTOMATION CONTROLLER AFTER INSTALLATION

After you install automation controller, you must log in.

Procedure

1. With the login information provided after your installation completed, open a web browser and log in to the automation controller by navigating to its server URL at: https://<CONTROLLER_SERVER_NAME>/

2. Use the credentials specified during the installation process to login:

   - The default username is **admin**.

   - The password for **admin** is the value specified.

3. Click the **More Actions** icon ⋮ next to the desired user.

4. Click **Edit**.

5. Edit the required details and click **Save**.

CHAPTER 3. THE USER INTERFACE

The Automation Execution *User Interface* (UI) provides a graphical framework for your IT orchestration requirements.

Access your user profile, the **About** page, view related documentation, or log out using the icons in the page header.

The navigation panel provides quick access to automation controller resources, such as **Jobs**, **Templates**, **Schedules**, **Projects**, **Infrastructure**, and **Administration**.

- Jobs
- Job templates
- Workflow job templates
- Schedules
- Projects

3.1. INFRASTRUCTURE MENU

The **Infrastucture** menu provides quick access to the following automation controller resources:

- Topology View
- Inventories
- Hosts
- Instance Groups
- Instances
- Execution Environments
- Credentials
- Credential Types

3.2. ADMINISTRATION

The **Administration** menu provides access to the administrative options of automation controller. From here, you can create, view, and edit:

- Activity Stream
- Workflow Approvals
- Notifiers
- Management Jobs

3.3. THE SETTINGS MENU

You can configure some automation controller options by using the **Settings** menu of the User Interface.

The **Settings** page enables an administrator to configure the following:

- Configuring Subscriptions
- Platform gateway
- User Preferences
- Configuring jobs
- Setting up logging
- Troubleshooting options

CHAPTER 4. SEARCH

Use automation controller’s search tool for search and filter capabilities across many functions. An expandable list of search conditions is available from the **Name** menu in the search field.

4.1. RULES FOR SEARCHING

These searching tips assume that you are not searching hosts.

- The typical syntax of a search consists of a field, followed by a value.
- A colon is used to separate the field that you want to search from the value.
- If the search has no colon (see example 3) it is treated as a simple string search where **?search=foobar** is sent.

The following are examples of syntax used for searching:

1. **name:localhost** In this example, the user is searching for the string **localhost** in the name attribute. If that string does not match something from **Fields** or **Related Fields**, the entire search is treated as a string.
2. **organization.name:Default** This example shows a Related Field Search. The period in **organization.name** separates the model from the field. Depending on how deep or complex the search is, you can have multiple periods in that part of the query.
3. **foobar** This is a simple string (key term) search that finds all instances of the search term using an **icontains** search against the name and description fields. If you use a space between terms, for example **foo bar**, then results that contain both terms are returned. If the terms are wrapped in quotes, for example, **"foo bar"**, automation controller searches for the string with the terms appearing together.
   Specific name searches search against the API name. For example, **Management job** in the user interface is **system_job** in the API.
4. **organization:Default** This example shows a Related Field search but without specifying a field to go along with the organization. This is supported by the API and is analogous to a simple string search but carried out against the organization (does an **icontains** search against both the name and description).

4.1.1. Values for search fields

To find values for certain fields, refer to the API endpoint for extensive options and their valid values. For example, if you want to search against **/api/v2/jobs** > **type** field, you can find the values by performing an **OPTIONS** request to **/api/v2/jobs** and look for entries in the API for **"type"**. Additionally, you can view the related searches by scrolling to the bottom of each screen. In the example for **/api/v2/jobs**, the related search shows:

```
"related_search_fields": [
        "modified_by__search",
        "project__search",
        "project_update__search",
        "credentials__search",
        "unified_job_template__search",
        "created_by__search",
        "inventory__search",
        "labels__search",
        "schedule__search",
        "webhook_credential__search",
        "job_template__search",
        "job_events__search",
        "dependent_jobs__search",
        "launch_config__search",
        "unifiedjob_ptr__search",
        "notifications__search",
        "unified_job_node__search",
        "instance_group__search",
        "hosts__search",
        "job_host_summaries__search"
```

The values for Fields come from the keys in a **GET** request. **url**, **related**, and **summary_fields** are not used. The values for Related Fields also come from the **OPTIONS** response, but from a different attribute. Related Fields is populated by taking all the values from **related_search_fields** and stripping off the **__search** from the end.

Any search that does not start with a value from Fields or a value from the Related Fields, is treated as a generic string search. Searching for **localhost**, for example, results in the UI sending **?search=localhost** as a query parameter to the API endpoint. This is a shortcut for an **icontains** search on the name and description fields.

4.1.2. Searching using values from related fields

Searching a Related Field requires you to start the search string with the Related Field. The following example describes how to search using values from the Related Field, *organization*.

The left-hand side of the search string must start with **organization**, for example, **organization:Default**. Depending on the related field, you can provide more specific direction for the search by providing secondary and tertiary fields. An example of this is to specify that you want to search for all job templates that use a project matching a certain name. The syntax on this would look like: **job_template.project.name:"A Project"**.

NOTE

This query executes against the **unified_job_templates** endpoint which is why it starts with **job_template**. If you were searching against the **job_templates** endpoint, then you would not need the **job_template** portion of the query.

4.1.3. Other search considerations

Be aware of the following issues when searching in automation controller:

- There is currently no supported syntax for **OR** queries. All search terms are **AND**ed in the query parameters.

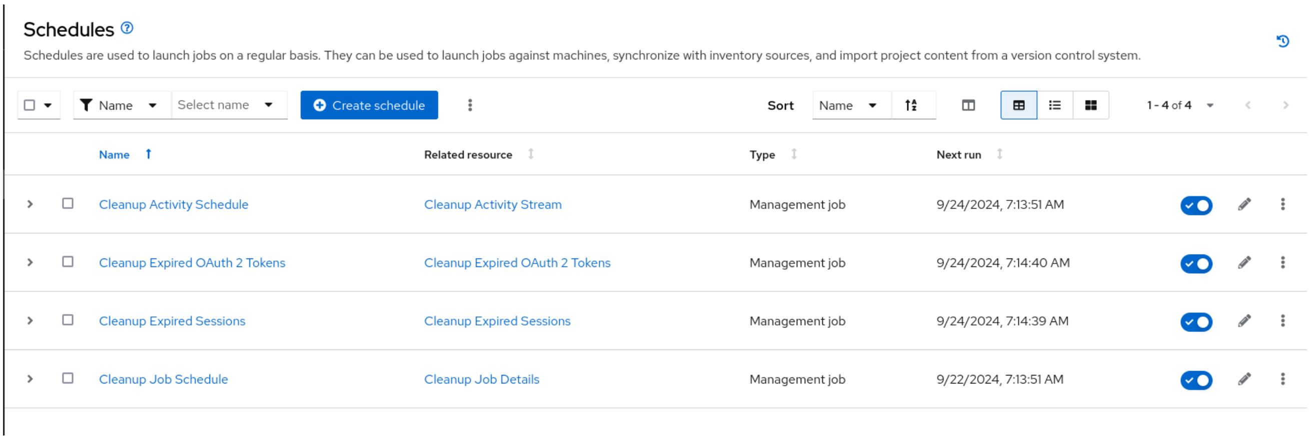
- The left part of a search parameter can be wrapped in quotes to support searching for strings with spaces. For more information, see Rules for searching.

- Currently, the values in the Fields are direct attributes expected to be returned in a **GET** request. Whenever you search against one of the values, automation controller carries out an **__icontains** search. So, for example, **name:localhost** sends back **?**

name__icontains=localhost. Automation controller currently performs this search for every Field value, even **id**.

4.2. SORT

Where applicable, use the arrows in each column to sort by ascending order. The following is an example from the schedules list:



Schedules

Schedules are used to launch jobs on a regular basis. They can be used to launch jobs against machines, synchronize with inventory sources, and import project content from a version control system.

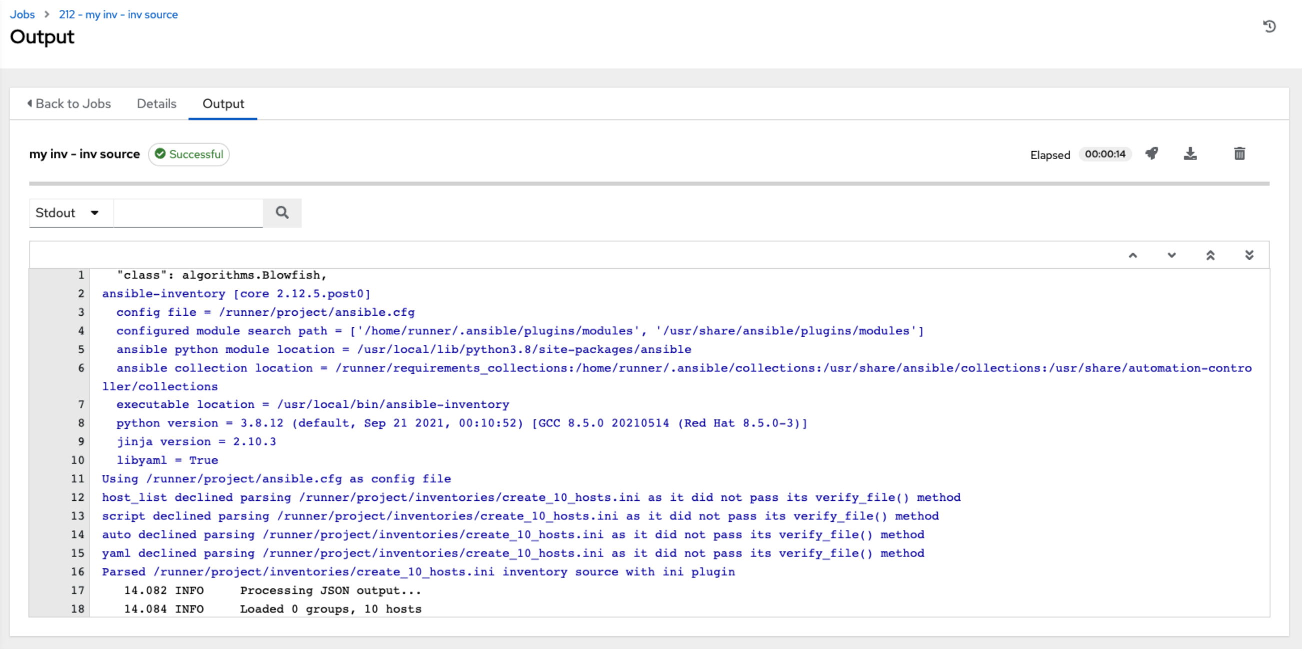
Name ↑	Related resource	Type	Next run
Cleanup Activity Schedule	Cleanup Activity Stream	Management job	9/24/2024, 7:13:51 AM
Cleanup Expired OAuth 2 Tokens	Cleanup Expired OAuth 2 Tokens	Management job	9/24/2024, 7:14:40 AM
Cleanup Expired Sessions	Cleanup Expired Sessions	Management job	9/24/2024, 7:14:39 AM
Cleanup Job Schedule	Cleanup Job Details	Management job	9/22/2024, 7:13:51 AM

The direction of the arrow indicates the sort order of the column.

CHAPTER 5. JOBS IN AUTOMATION CONTROLLER

A job is an instance of automation controller launching an Ansible Playbook against an inventory of hosts.

The **Jobs** list view displays a list of jobs and their statuses, shown as completed successfully, failed, or as an active (running) job. The default view is collapsed (Compact) with the job name, status, job type, start, and finish times. You can click the arrow > icon to expand and see more information. You can sort this list by various criteria, and perform a search to filter the jobs of interest.

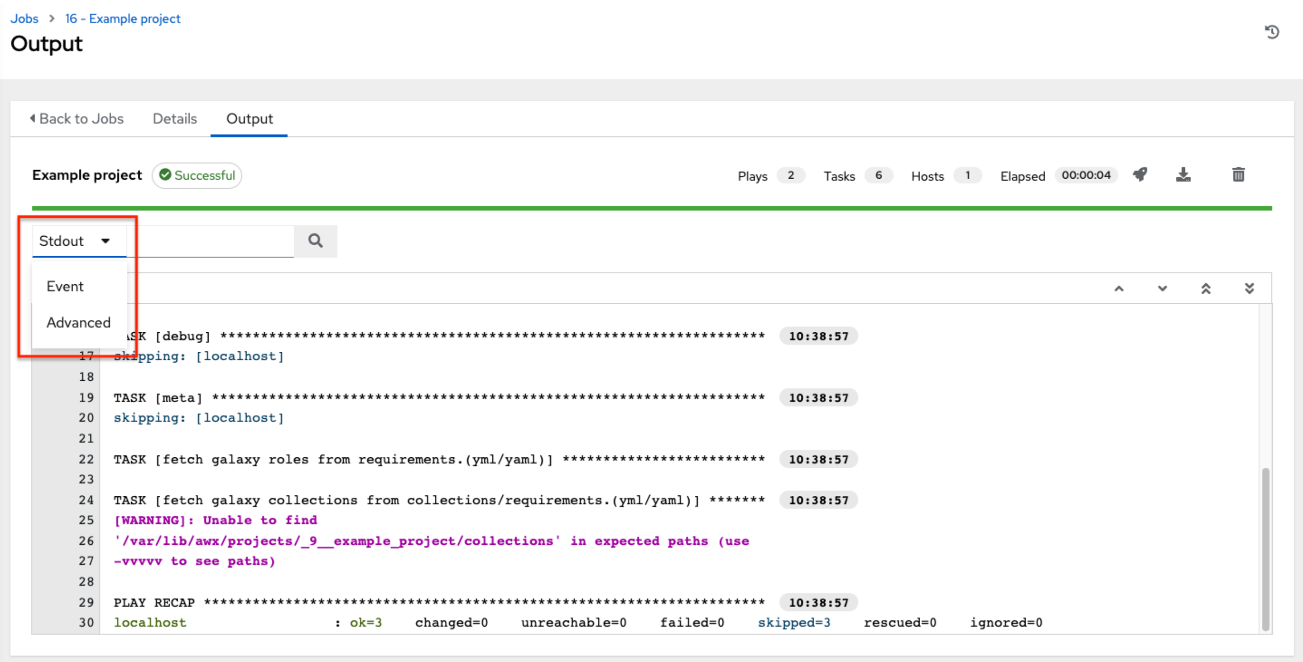


```
  "class": algorithms.Blowfish,
ansible-inventory [core 2.12.5.post0]
  config file = /runner/project/ansible.cfg
  configured module search path = ['/home/runner/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/local/lib/python3.8/site-packages/ansible
  ansible collection location = /runner/requirements_collections:/home/runner/.ansible/collections:/usr/share/ansible/collections:/usr/share/automation-contro
ller/collections
  executable location = /usr/local/bin/ansible-inventory
  python version = 3.8.12 (default, Sep 21 2021, 00:10:52) [GCC 8.5.0 20210514 (Red Hat 8.5.0-3)]
  jinja version = 2.10.3
  libyaml = True
Using /runner/project/ansible.cfg as config file
host_list declined parsing /runner/project/inventories/create_10_hosts.ini as it did not pass its verify_file() method
script declined parsing /runner/project/inventories/create_10_hosts.ini as it did not pass its verify_file() method
auto declined parsing /runner/project/inventories/create_10_hosts.ini as it did not pass its verify_file() method
yaml declined parsing /runner/project/inventories/create_10_hosts.ini as it did not pass its verify_file() method
Parsed /runner/project/inventories/create_10_hosts.ini inventory source with ini plugin
   14.082 INFO     Processing JSON output...
   14.084 INFO     Loaded 0 groups, 10 hosts
```

From this screen you can complete the following tasks:

- View details and standard output of a particular job
- Relaunch 🚀 jobs
- Cancel or Remove selected jobs

The relaunch operation only applies to relaunches of playbook runs and does not apply to project or inventory updates, system jobs, and workflow jobs. When a job relaunches, the **Jobs Output** view is displayed. Selecting any type of job also takes you to the **Job Output** view for that job, where you can filter jobs by various criteria:



```
TASK [debug] *******************************************************************   10:38:57
skipping: [localhost]

TASK [meta] ********************************************************************   10:38:57
skipping: [localhost]

TASK [fetch galaxy roles from requirements.(yml/yaml)] *************************   10:38:57

TASK [fetch galaxy collections from collections/requirements.(yml/yaml)] *******   10:38:57
[WARNING]: Unable to find
'/var/lib/awx/projects/_9__example_project/collections' in expected paths (use
-vvvvv to see paths)

PLAY RECAP *********************************************************************   10:38:57
localhost                  : ok=3    changed=0    unreachable=0    failed=0    skipped=3    rescued=0    ignored=0
```

- The **Event** option in the **Search output** list enables you to filter by the events of interest, such as errors, host failures, host retries, and items skipped. You can include as many events in the filter as necessary. For more information about using the search, see the Search section.

5.1. INVENTORY SYNC JOBS

When an inventory synchronization is executed, the results display in the **Output** tab.

For more information about inventory synchronization, see Constructed inventories.

If used, the Ansible CLI displays the same information. This can be useful for debugging. The **ANSIBLE_DISPLAY_ARGS_TO_STDOUT** parameter is set to **False** for all playbook runs. This parameter matches Ansible's default behavior and does not display task arguments in task headers in the **Job Detail** interface to avoid leaking certain sensitive module parameters to **stdout**. To restore the earlier behavior, set **ANSIBLE_DISPLAY_ARGS_TO_STDOUT** to **True** through the **AWX_TASK_ENV** configuration setting.

For more information, see ANSIBLE_DISPLAY_ARGS_TO_STDOUT in the ansible documentation.

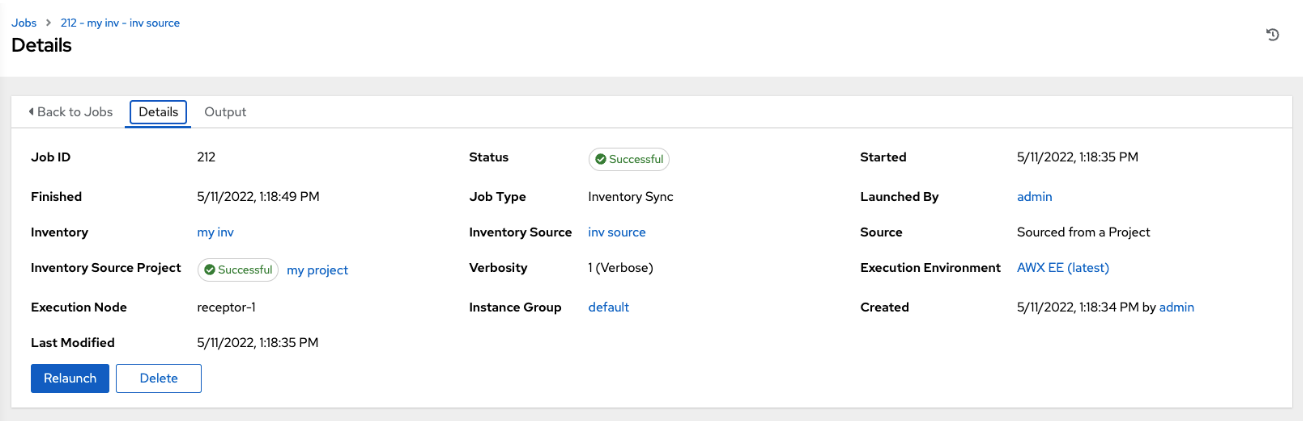
You can **Relaunch job**, **Cancel job**, download the job output, or delete the job.

NOTE

You can perform an inventory update while a related job is running. In cases where you have a large project (around 10 GB), disk space on **/tmp** can be an issue.

5.1.1. Inventory sync details

Access the **Details** tab to view details about the job execution:



Jobs > 212 - my inv - inv source

Details

Back to Jobs | Details | Output

Job ID	212	Status	✓ Successful	Started	5/11/2022, 1:18:35 PM
Finished	5/11/2022, 1:18:49 PM	Job Type	Inventory Sync	Launched By	admin
Inventory	my inv	Inventory Source	inv source	Source	Sourced from a Project
Inventory Source Project	✓ Successful my project	Verbosity	1 (Verbose)	Execution Environment	AWX EE (latest)
Execution Node	receptor-1	Instance Group	default	Created	5/11/2022, 1:18:34 PM by admin
Last Modified	5/11/2022, 1:18:35 PM				

Relaunch | Delete

You can view the following details for an executed job:

- **Status**: It can be any of the following:
  - **Pending**: The inventory sync has been created, but not queued or started yet. Any job, not just inventory source syncs, stays in pending until it is ready to be run by the system. Reasons for inventory source syncs not being ready include:
    - Dependencies that are currently running (all dependencies must be completed before the next step can execute).
    - Insufficient capacity to run in the locations it is configured for.
  - **Waiting**: The inventory sync is in the queue waiting to be executed.
  - **Running**: The inventory sync is currently in progress.
  - **Successful**: The inventory sync job succeeded.
  - **Failed**: The inventory sync job failed.
- **Inventory**: The name of the associated inventory group.
- **Source**: The type of cloud inventory.
- **Inventory Source Project** The project used as the source of this inventory sync job.
- **Execution Environment**: The execution environment used.
- **Execution node**: The node used to execute the job.
- **Instance Group**: The name of the instance group used with this job (automation controller is the default instance group).

Selecting these items enables you to view the corresponding job templates, projects, and other objects.

5.2. SCM INVENTORY JOBS

When an inventory sourced from an SCM, for example git, is executed, the results are displayed in the **Output** tab. If used, the Ansible CLI displays the same information. This can be useful for debugging. Use the navigation menu to **Relaunch job**, **Cancel job**, download ⬇ the job output, or delete 🗑 the job.

5.2.1. SCM inventory details

To view details about the job execution and its associated project, select the **Details** tab.

You can view the following details for an executed job:

- **Status**: It can be any of the following:
  - **Pending**: The SCM job has been created, but not queued or started yet. Any job, not just SCM jobs, stay in pending until it is ready to be run by the system. Reasons for SCM jobs not being ready include dependencies that are currently running (all dependencies must be completed before the next step can execute), or there is not enough capacity to run in the locations it is configured to.
  - **Waiting**: The SCM job is in the queue waiting to be executed.
  - **Running**: The SCM job is currently in progress.
  - **Successful**: The last SCM job succeeded.
  - **Failed**: The last SCM job failed.
- **Job Type**: SCM jobs display Source Control Update.
- **Project**: The name of the project.
- **Project Status**: Indicates whether the associated project was successfully updated.
- **Revision**: Indicates the revision number of the sourced project that was used in this job.
- **Execution Environment**: Specifies the execution environment used to run this job.
- **Execution Node**: Indicates the node on which the job ran.
- **Instance Group**: Indicates the instance group on which the job ran, if specified.
- **Job Tags**: Tags show the various job operations executed.

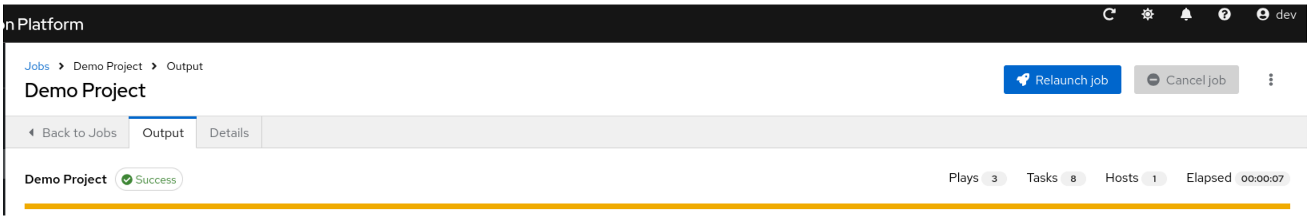
Select these items to view the corresponding job templates, projects, and other objects.

5.3. PLAYBOOK RUN JOBS

When a playbook is executed, the results display in the **Output** tab. If used, the Ansible CLI displays the same information. This can be useful for debugging.

The events summary displays the following events that are run as part of this playbook:

- The number of times this playbook has run is shown in the **Plays** field
- The number of tasks associated with this playbook is shown in the **Tasks** field
- The number of hosts associated with this playbook is shown in the **Hosts** field
- The amount of time it took to complete the playbook run is shown in the **Elapsed** field



You can **Relaunch job**, **Cancel job**, download ⬇ the job output, or delete 🗑 the job.

Hover over a section of the host status bar in the **Output** view and the number of hosts associated with that status displays.

The output for a playbook job is also available after launching a job from the **Jobs** tab of its **Jobs Templates** page. View its host details by clicking the line item tasks in the output.

5.3.1. Search

Use **Search** to look up specific events, hostnames, and their statuses. To filter only certain hosts with a particular status, specify one of the following valid statuses:

ok

Indicates that a task completed successfully but no change was executed on the host.

changed

The playbook task executed. Since Ansible tasks should be written to be idempotent, tasks can exit successfully without executing anything on the host. In these cases, the task returns **ok**, but not **changed**.

failed

The task failed. Further playbook execution stopped for this host.

unreachable

The host is unreachable from the network or has another unrecoverable error associated with it.

skipped

The playbook task skipped because no change was necessary for the host to reach the target state.

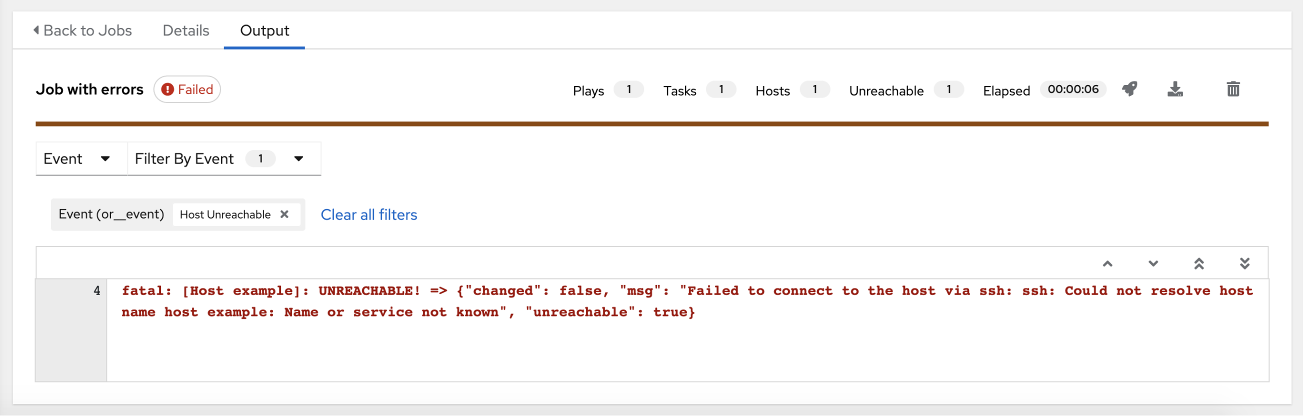
rescued

This shows the tasks that failed and then executes a rescue section.

ignored

This shows the tasks that failed and have **ignore_errors: yes configured**.

The following example shows a search with only unreachable hosts:

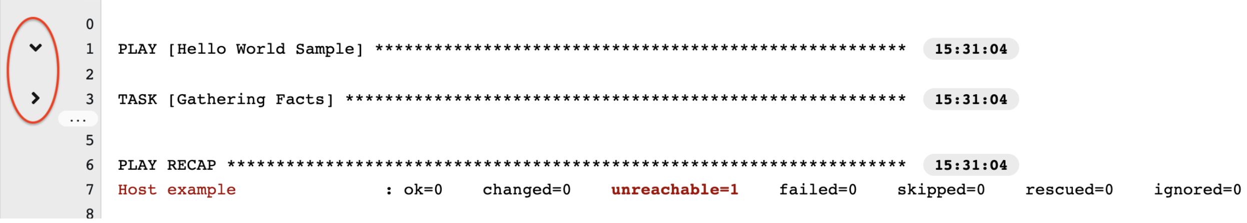


```
4  fatal: [Host example]: UNREACHABLE! => {"changed": false, "msg": "Failed to connect to the host via ssh: ssh: Could not resolve host
   name host example: Name or service not known", "unreachable": true}
```

For more information on using the search, see the Search section.

The standard output view displays the events that occur on a particular job. By default, all rows are expanded so that the details are displayed. Use the collapse-all (≡) icon to switch to a view that only contains the headers for plays and tasks. Click the plus (+) icon to view all the lines of the standard output.

You can display all the details of a specific play or task by clicking the arrow icons next to them. Click an arrow from sideways to downward to expand the lines associated with that play or task. Click the arrow back to the sideways position to collapse and hide the lines.



```
PLAY [Hello World Sample] *****************************************************   15:31:04

TASK [Gathering Facts] ********************************************************   15:31:04
...

PLAY RECAP ********************************************************************   15:31:04
Host example              : ok=0    changed=0    unreachable=1    failed=0    skipped=0    rescued=0    ignored=0
```

When viewing details in the expand or collapse mode, note the following:

- Each displayed line that is not collapsed has a corresponding line number and start time.
- An expand or collapse icon is at the start of any play or task after the play or task has completed.
- If querying for a particular play or task, it appears collapsed at the end of its completed process.
- In some cases, an error message appears, stating that the output may be too large to display. This occurs when there are more than 4000 events. Use the search and filter for specific events to bypass the error.

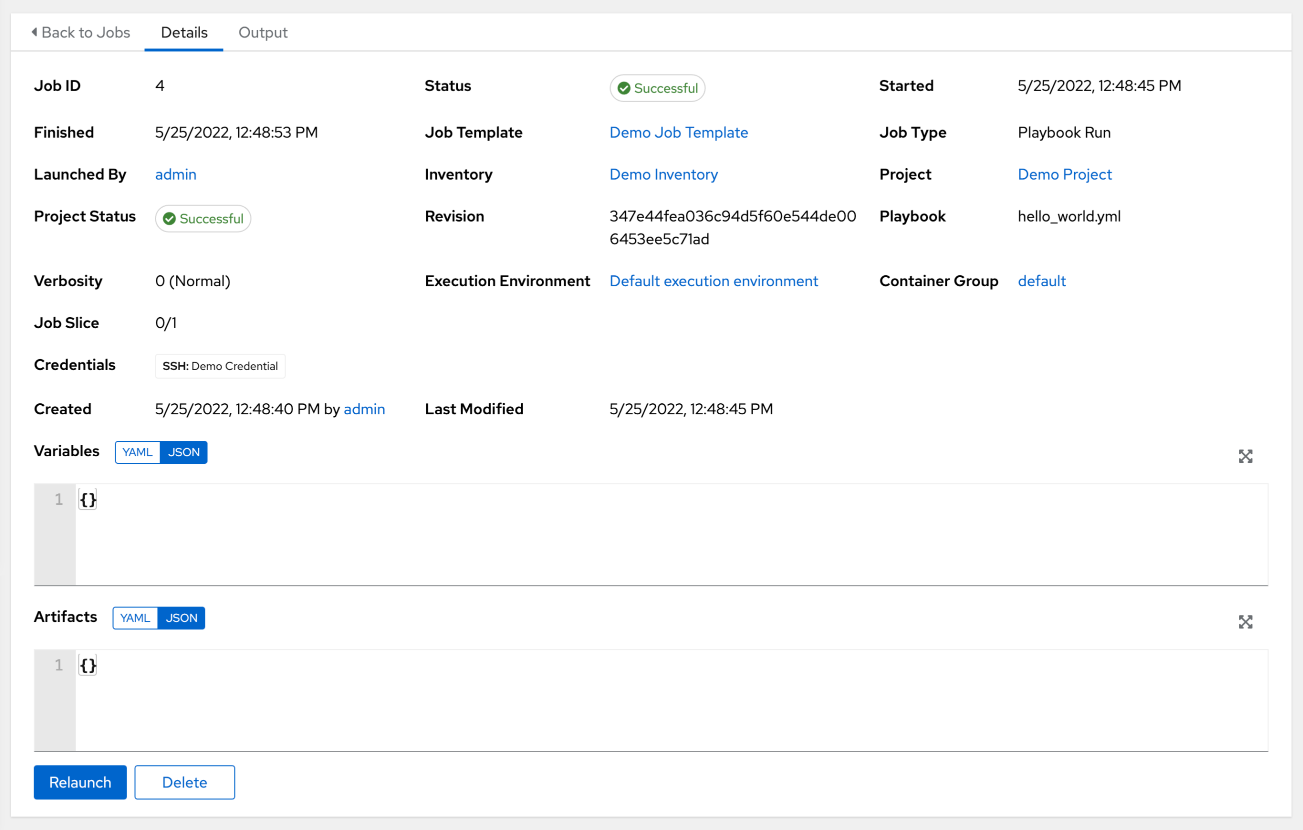
Click on a line of an event from the **Stdout** pane and a **Host Events** window displays in a separate window. This window shows the host that was affected by that particular event.

NOTE

Upgrading to the latest versions of Ansible Automation Platform involves progressively migrating all historical playbook output and events. This migration process is gradual, and happens automatically in the background after installation is complete. Installations with very large amounts of historical job output (tens or hundreds of GB of output) can have missing job output until migration is complete. The most recent data shows up at the top of the output, followed by older events.

5.3.2. Playbook run details

Access the **Details** tab to view details about the job execution:



Job ID	4	Status	Successful	Started	5/25/2022, 12:48:45 PM
Finished	5/25/2022, 12:48:53 PM	Job Template	Demo Job Template	Job Type	Playbook Run
Launched By	admin	Inventory	Demo Inventory	Project	Demo Project
Project Status	Successful	Revision	347e44fea036c94d5f60e544de00 6453ee5c71ad	Playbook	hello_world.yml
Verbosity	0 (Normal)	Execution Environment	Default execution environment	Container Group	default
Job Slice	0/1				
Credentials	SSH: Demo Credential				
Created	5/25/2022, 12:48:40 PM by admin	Last Modified	5/25/2022, 12:48:45 PM		

Variables

```
{}
```

Artifacts

```
{}
```

You can view the following details for an executed job:

- **Status**: It can be any of the following:
  - **Pending**: The playbook run has been created, but not queued or started yet. Any job, not just playbook runs, stay in pending until it is ready to be run by the system. Reasons for playbook runs not being ready include dependencies that are currently running (all dependencies must be completed before the next step can execute), or there is not enough capacity to run in the locations it is configured to.
  - **Waiting**: The playbook run is in the queue waiting to be executed.
  - **Running**: The playbook run is currently in progress.
  - **Successful**: The last playbook run succeeded.
  - **Failed**: The last playbook run failed.
- **Job Template**: The name of the job template from which this job launched.
- **Inventory**: The inventory selected to run this job against.
- **Project**: The name of the project associated with the launched job.
- **Project Status**: The status of the project associated with the launched job.
- **Playbook**: The playbook used to launch this job.
- **Execution Environment**: The name of the execution environment used in this job.
- **Container Group**: The name of the container group used in this job.
- **Credentials**: The credentials used in this job.

- **Extra Variables**: Any extra variables passed when creating the job template are displayed here.

Select one of these items to view the corresponding job templates, projects, and other objects.

5.3.3. Playbook Access and Information Sharing

Automation controller's use of automation execution environments and Linux containers prevents playbooks from reading files outside of their project directory.

By default, the only data exposed to the ansible-playbook process inside the container is the current project being used.

You can customize this in the Job Settings and expose additional directories from the host into the container.

5.3.4. 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.

If you need to expose additional directories, you must customize your playbook runs. To configure job isolation, you can set variables.

By default, automation controller uses 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** page, or in the REST API at **/api/v2/settings/jobs**:

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

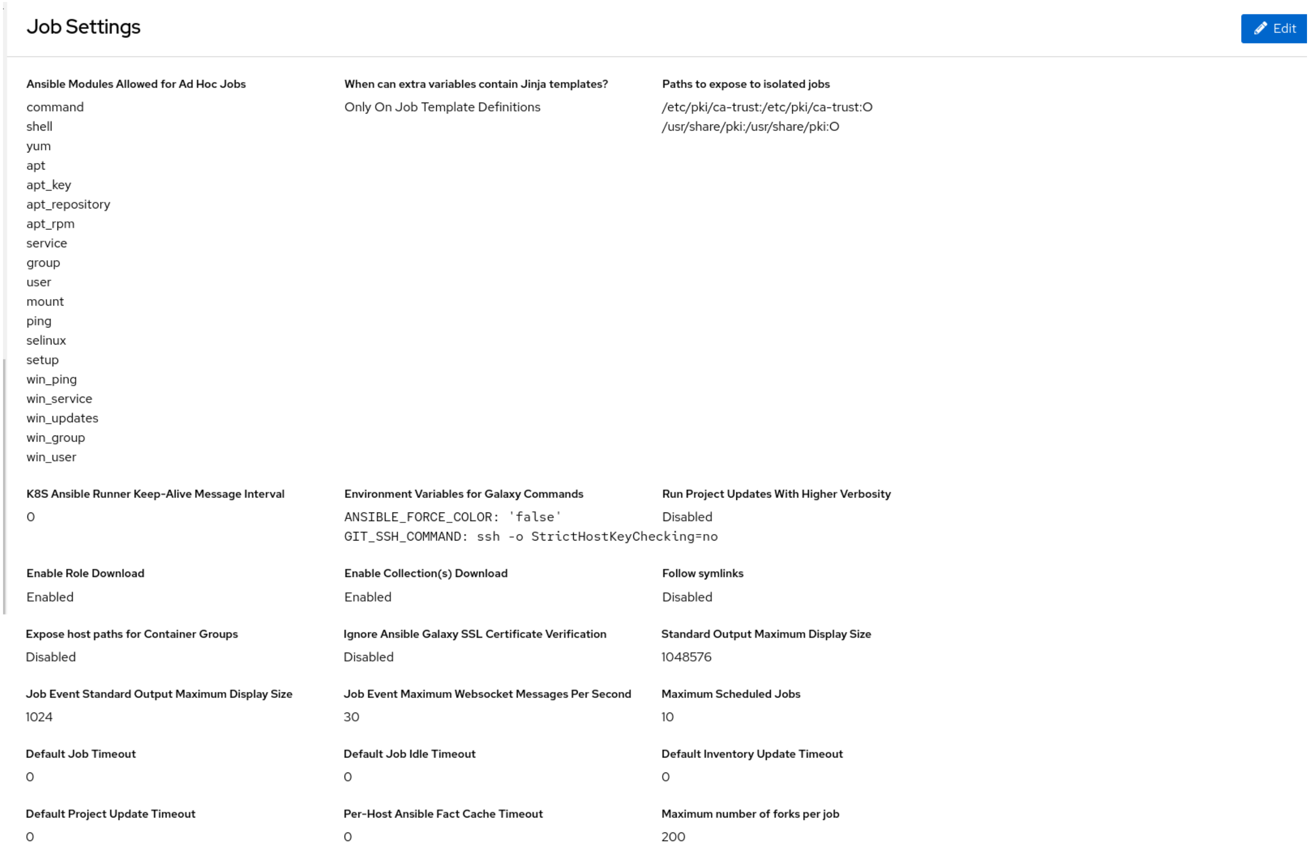
If there are any additional directories that should specifically be exposed from the host to the container that playbooks run in, you can specify those in the **Paths to expose to isolated jobs**field of the **Jobs Settings** page, or in the REST API at **/api/v2/settings/jobs**:

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

NOTE

If your playbooks need to use keys or settings defined in **AWX_ISOLATION_SHOW_PATHS**, then add this file to **/var/lib/awx/.ssh**.

The fields described here can be found on the **Jobs settings** page:



Job Settings

Ansible Modules Allowed for Ad Hoc Jobs
command
shell
yum
apt
apt_key
apt_repository
apt_rpm
service
group
user
mount
ping
selinux
setup
win_ping
win_service
win_updates
win_group
win_user

When can extra variables contain Jinja templates?
Only On Job Template Definitions

Paths to expose to isolated jobs
/etc/pki/ca-trust:/etc/pki/ca-trust:O
/usr/share/pki:/usr/share/pki:O

K8S Ansible Runner Keep-Alive Message Interval
0

Environment Variables for Galaxy Commands

```
ANSIBLE_FORCE_COLOR: 'false'
GIT_SSH_COMMAND: ssh -o StrictHostKeyChecking=no
```

Run Project Updates With Higher Verbosity
Disabled

Enable Role Download
Enabled

Enable Collection(s) Download
Enabled

Follow symlinks
Disabled

Expose host paths for Container Groups
Disabled

Ignore Ansible Galaxy SSL Certificate Verification
Disabled

Standard Output Maximum Display Size
1048576

Job Event Standard Output Maximum Display Size
1024

Job Event Maximum Websocket Messages Per Second
30

Maximum Scheduled Jobs
10

Default Job Timeout
0

Default Job Idle Timeout
0

Default Inventory Update Timeout
0

Default Project Update Timeout
0

Per-Host Ansible Fact Cache Timeout
0

Maximum number of forks per job
200

5.4. AUTOMATION CONTROLLER CAPACITY DETERMINATION AND JOB IMPACT

The automation controller capacity system determines how many jobs can run on an instance given the amount of resources available to the instance and the size of the jobs that are running (referred to as Impact). The algorithm used to determine this is based on the following two things:

- How much memory is available to the system (**mem_capacity**)
- How much processing capacity is available to the system (**cpu_capacity**)

Capacity also impacts instance groups. Since groups are made up of instances, instances can also be assigned to multiple groups. This means that impact to one instance can affect the overall capacity of other groups.

Instance groups, not instances themselves, can be assigned to be used by jobs at various levels. For more information, see Clustering in *Configuring automation execution*.

When the Task Manager prepares its graph to determine which group a job runs on, it commits the capacity of an instance group to a job that is not ready to start yet.

In smaller configurations, if only one instance is available for a job to run, the Task Manager enables that job to run on the instance even if it pushes the instance over capacity. This guarantees that jobs do not get stuck as a result of an under-provisioned system.

Additional resources

- For information about container groups, see Capacity settings for instance group and container group in *Configuring automation execution*.

- For information about sliced jobs and their impact to capacity, see Job slice execution behavior.

5.4.1. Resource determination for capacity algorithm

Capacity algorithms determine how many forks a system is capable of running simultaneously. These algorithms control how many systems Ansible can communicate with simultaneously. Increasing the number of forks an automation controller system is running enables jobs to run faster by performing more work in parallel. However, this increases the load on the system, which can cause work to slow down.

The default, **mem_capacity**, enables you to over-commit processing resources while protecting the system from running out of memory. If most of your work is not processor-bound, then selecting this mode maximizes the number of forks.

5.4.1.1. Memory relative capacity

mem_capacity is calculated relative to the amount of memory needed per fork. Taking into account the overhead for internal components, this is approximately 100MB per fork. When considering the amount of memory available to Ansible jobs, the capacity algorithm reserves 2GB of memory to account for the presence of other services. The algorithm formula for this is:

```
(mem - 2048) / mem_per_fork
```

The following is an example:

```
(4096 - 2048) / 100 == ~20
```

A system with 4GB of memory is capable of running 20 forks. The value **mem_per_fork** is controlled by setting the value of **SYSTEM_TASK_FORKS_MEM**, which defaults to 100.

5.4.1.2. CPU relative capacity

Ansible workloads are often processor-bound. In such cases, you can reduce the simultaneous workload to enable more tasks to run faster and reduce the average time-to-completion of those jobs.

Just as the **mem_capacity** algorithm adjusts the amount of memory required per fork, the **cpu_capacity** algorithm adjusts the amount of processing resources required per fork. The baseline value for this is four forks per core. The algorithm formula for this is:

```
cpus * fork_per_cpu
```

For example, a 4-core system looks like the following:

```
4 * 4 == 16
```

You can control the value of **fork_per_cpu** by setting the value of **SYSTEM_TASK_FORKS_CPU** which defaults to 4.

5.4.2. Capacity job impacts

When selecting the capacity, it is important to understand how each job type affects capacity.

The default forks value for Ansible is five. However, if you set up automation controller to run against fewer systems than that, then the actual concurrency value is lower.

When a job is run in automation controller, the number of forks selected is incremented by 1, to compensate for the Ansible parent process.

Example

If you run a playbook against five systems with forks value of 5, then the actual forks value from the Job Impact perspective is 6.

5.4.2.1. Impact of job types in automation controller

Jobs and ad hoc jobs follow the preceding model, forks +1. If you set a fork value on your job template, your job capacity value is the minimum of the forks value supplied and the number of hosts that you have, plus one. The +1 is to account for the parent Ansible process.

Instance capacity determines which jobs get assigned to any specific instance. Jobs and ad hoc commands use more capacity if they have a higher forks value.

Job types including the following, have a fixed impact:

- Inventory updates: 1
- Project updates: 1
- System jobs: 5

NOTE

If you do not set a forks value on your job template, your job uses Ansible’s default forks value of five. However, it uses fewer if your job has fewer than five hosts. In general, setting a forks value higher than what the system is capable of can cause issues by running out of memory or over-committing CPU. The job template fork values that you use must fit on the system. If you have playbooks using 1000 forks but none of your systems individually has that much capacity, then your systems are undersized and at risk of performance or resource issues.

5.4.2.2. Selecting the correct capacity

Selecting a capacity out of the CPU-bound or the memory-bound capacity limits is selecting between the minimum or maximum number of forks. In the previous examples, the CPU capacity permits a maximum of 16 forks while the memory capacity permits 20. For some systems, the disparity between these can be large and you might want to have a balance between these two.

The instance field **capacity_adjustment** enables you to select how much you want to consider. It is represented as a value between 0.0 and 1.0. If set to a value of 1.0, then the largest value is used. The previous example involves memory capacity, so a value of 20 forks can be selected. If set to a value of 0.0 then the smallest value is used. A value of 0.5 is a 50/50 balance between the two algorithms, which is 18:

```
16 + (20 - 16) * 0.5 = 18
```

Procedure

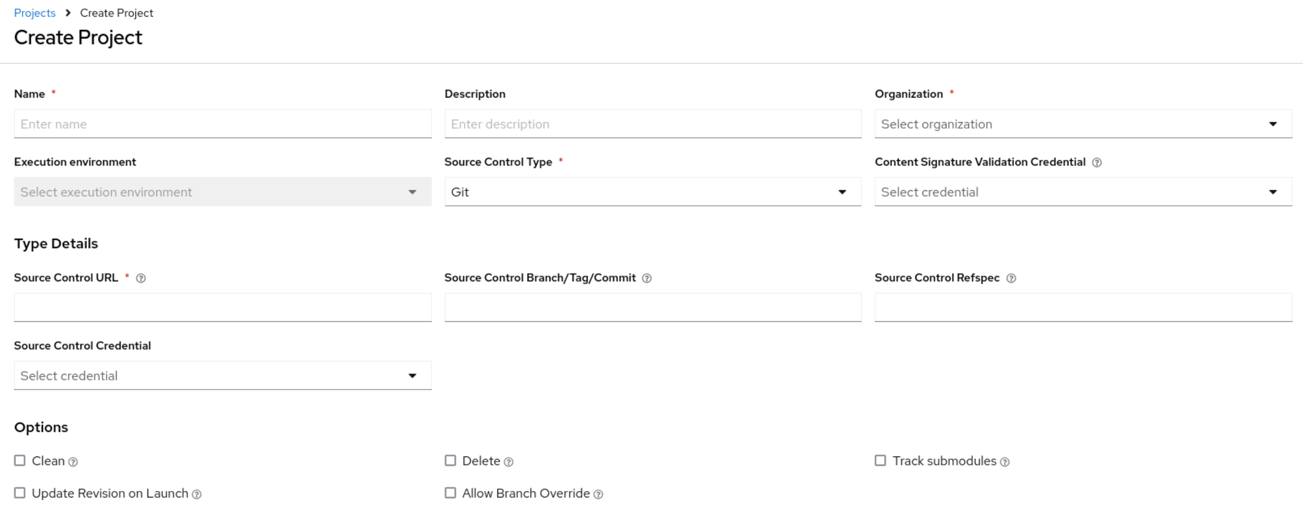
View or edit the capacity:

1. From the navigation panel, select **Automation Execution** → **Infrastructure** → **Instance Groups**.
2. On the **Instance Groups** list view, select the required instance.
3. Select the **Instances** tab and adjust the **Capacity adjustment** slider.

> **NOTE**
>
> The slider adjusts whether the instance capacity algorithm yields less forks (towards the left) or yields more forks (towards the right).

5.5. JOB BRANCH OVERRIDING

Projects specify the branch, tag, or reference to use from source control in the **scm_branch** field. These are represented by the values specified in the **Type Details** fields:



Projects › Create Project

Create Project

Name * : Enter name

Description: Enter description

Organization * : Select organization

Execution environment: Select execution environment

Source Control Type * : Git

Content Signature Validation Credential: Select credential

Type Details

Source Control URL *

Source Control Branch/Tag/Commit

Source Control Refspec

Source Control Credential: Select credential

Options

☐ Clean

☐ Delete

☐ Track submodules

☐ Update Revision on Launch

☐ Allow Branch Override

When creating or editing a job you have the option to **Allow branch override**. When this option is checked, project administrators can delegate branch selection to the job templates that use that project, requiring only project **use_role**.

5.5.1. Source tree copy behavior

Every job run has its own private data directory. This directory contains a copy of the project source tree for the given **scm_branch** that the job is running. Jobs are free to make changes to the project folder and make use of those changes while it is still running. This folder is temporary and is removed at the end of the job run.

If you check the **Clean** option, modified files are removed in automation controller's local copy of the repository. This is done through use of the force parameter in its corresponding Ansible modules pertaining to git or Subversion.

Additional resources

For more information, see the Parameters section of the Ansible documentation.

5.5.2. Project revision behavior

During a project update, the revision of the default branch (specified in the **Source control branch** field of the project) is stored when updated. If providing a non-default **Source control branch** (not a commit hash or tag) in a job, the newest revision is pulled from the source control remote immediately before the job starts. This revision is shown in the **Source control revision** field of the job and its project update.

As a result, offline job runs are impossible for non-default branches. To ensure that a job is running a static version from source control, use tags or commit hashes. Project updates do not save all branches, only the project default branch.

The **Source control branch** field is not validated, so the project must update to assure it is valid. If this field is provided or prompted for, the **Playbook** field of job templates is not validated, and you have to launch the job template to verify presence of the expected playbook.

5.5.3. Git Refspec

The **Source control refspec** field specifies which extra references the update should download from the remote. Examples include the following:

- **refs/:refs/remotes/origin/**: This fetches all references, including remotes of the remote
- **refs/pull/:refs/remotes/origin/pull/** (GitHub-specific): This fetches all refs for all pull requests
- **refs/pull/62/head:refs/remotes/origin/pull/62/head**: This fetches the ref for one GitHub pull request

For large projects, consider performance impact when using the first or second examples.

The **Source control refspec** parameter affects the availability of the project branch, and can enable access to references not otherwise available. Use the earlier examples to supply a pull request from the **Source control branch**, which is not possible without the **Source control refspec** field.

The Ansible git module fetches **refs/heads/** by default. This means that you can use a project's branches, tags and commit hashes, as the **Source control branch** if **Source control refspec** is blank. The value specified in the **Source control refspec** field affects which **Source control branch** fields can be used as overrides. Project updates (of any type) perform an extra **git fetch** command to pull that refspec from the remote.

Example

You can set up a project that enables branch override with the first or second refspec example. Use this in a job template that prompts for the **Source control branch**. A client can then launch the job template when a new pull request is created, providing the branch **pull/N/head** and the job template can run against the provided GitHub pull request reference.

Additional resources

For more information, see the Ansible git module .

CHAPTER 6. JOB TEMPLATES

You can create both Job templates and Workflow job templates from **Automation Execution → Templates**.

For Workflow job templates, see Workflow job templates.

A job template is a definition and set of parameters for running an Ansible job. Job templates are useful to run the same job many times. They also encourage the reuse of Ansible Playbook content and collaboration between teams.

The **Templates** page shows both job templates and workflow job templates that are currently available. The default view is collapsed (Compact), showing the template name, template type, and the timestamp of the last job that ran using that template. You can click the arrow ❯ icon next to each entry to expand and view more information. This list is sorted alphabetically by name, but you can sort by other criteria, or search by various fields and attributes of a template.

From this screen you can launch 🚀 , edit ✎ , copy ⧉ and delete 🗑 a job template.

Workflow templates have the workflow visualizer ⚯ icon as a shortcut for accessing the workflow editor.

NOTE

You can use job templates to build a workflow template. Templates that show the **Workflow Visualizer** ⚯ icon next to them are workflow templates. Clicking the icon allows you to build a workflow graphically. Many parameters in a job template enable you to select **Prompt on Launch** that you can change at the workflow level, and do not affect the values assigned at the job template level. For instructions, see the Workflow Visualizer section.

6.1. CREATING A JOB TEMPLATE

Procedure

1. From the navigation panel, select **Automation Execution → Templates**.

2. On the **Templates** page, select **Create job template** from the **Create template** list.

3. Enter the appropriate details in the following fields:

   NOTE

   If a field has the **Prompt on launch** checkbox selected, launching the job prompts you for the value for that field when launching.

   Most prompted values override any values set in the job template.

   Exceptions are noted in the following table.

Field	Options	Prompt on Launch

Field	Options	Prompt on Launch
Name	Enter a name for the job.	N/A
Description	Enter an arbitrary description as appropriate (optional).	N/A
Job type	Choose a job type: - Run: Start the playbook when launched, running Ansible tasks on the selected hosts. - Check: Perform a "dry run" of the playbook and report changes that would be made without actually making them. Tasks that do not support check mode are missed and do not report potential changes. For more information about job types see the Playbooks section of the Ansible documentation.	Yes
Inventory	Choose the inventory to use with this job template from the inventories available to the logged in user. A System Administrator must grant you or your team permissions to be able to use certain inventories in a job template.	Yes. Inventory prompts show up as its own step in a later prompt window.
Project	Select the project to use with this job template from the projects available to the user that is logged in.	N/A

Field	Options	Prompt on Launch
Source control branch	This field is only present if you chose a project that allows branch override. Specify the overriding branch to use in your job run. If left blank, the specified SCM branch (or commit hash or tag) from the project is used. For more information, see Job branch overriding.	Yes
Execution Environment	Select the container image to be used to run this job. You must select a project before you can select an execution environment.	Yes. Execution environment prompts show up as its own step in a later prompt window.
Playbook	Choose the playbook to be launched with this job template from the available playbooks. This field automatically populates with the names of the playbooks found in the project base path for the selected project. Alternatively, you can enter the name of the playbook if it is not listed, such as the name of a file (such as foo.yml) you want to use to run with that playbook. If you enter a filename that is not valid, the template displays an error, or causes the job to fail.	N/A

Field	Options	Prompt on Launch
Credentials	Select the 🔍 icon to open a separate window. Choose the credential from the available options to use with this job template. Use the drop-down menu list to filter by credential type if the list is extensive. Some credential types are not listed because they do not apply to certain job templates.	● If selected, when launching a job template that has a default credential and supplying another credential replaces the default credential if it is the same type. The following is an example this message: **Job Template default credentials must be replaced with one of the same type. Please select a credential for the following types in order to proceed: Machine.** ● You can add more credentials as you see fit. ● Credential prompts show up as its own step in a later prompt window.

Field	Options	Prompt on Launch
Labels	• Optionally supply labels that describe this job template, such as **dev** or **test**. • Use labels to group and filter job templates and completed jobs in the display. • Labels are created when they are added to the job template. Labels are associated with a single Organization by using the Project that is provided in the job template. Members of the Organization can create labels on a job template if they have edit permissions (such as the admin role). • Once you save the job template, the labels appear in the **Job Templates** overview in the Expanded view. • Select ✕ beside a label to remove it. When a label is removed, it is no longer associated with that particular Job or Job Template, but it remains associated with any other jobs that reference it. • Jobs inherit labels from the Job Template at the time of launch. If you delete a label from a Job Template, it is also deleted from the Job.	• If selected, even if a default value is supplied, you are prompted when launching to supply additional labels, if needed. • You cannot delete existing labels, selecting ✕ only removes the newly added labels, not existing default labels.
Forks	The number of parallel or simultaneous processes to use while executing the playbook. A value of zero uses the Ansible default setting, which is five parallel processes unless overridden in **/etc/ansible/ansible.cfg**.	Yes

Field	Options	Prompt on Launch
Limit	A host pattern to further constrain the list of hosts managed or affected by the playbook. You can separate many patterns by colons (:). As with core Ansible: - a:b means "in group a or b" - a:b:&c means "in a or b but must be in c" - a:!b means "in a, and definitely not in b" For more information, see Patterns: targeting hosts and groups in the Ansible documentation.	Yes If not selected, the job template executes against all nodes in the inventory or only the nodes predefined on the **Limit** field. When running as part of a workflow, the workflow job template limit is used instead.
Verbosity	Control the level of output Ansible produces as the playbook executes. Choose the verbosity from Normal to various Verbose or Debug settings. This only appears in the **details** report view. Verbose logging includes the output of all commands. Debug logging is exceedingly verbose and includes information about SSH operations that can be useful in certain support instances. Verbosity **5** causes automation controller to block heavily when jobs are running, which could delay reporting that the job has finished (even though it has) and can cause the browser tab to lock up.	Yes
Job slicing	Specify the number of slices you want this job template to run. Each slice runs the same tasks against a part of the inventory. For more information about job slices, see Job Slicing.	Yes

Field	Options	Prompt on Launch
Timeout	This enables you to specify the length of time (in seconds) that the job can run before it is canceled. Consider the following for setting the timeout value: - There is a global timeout defined in the settings which defaults to 0, indicating no timeout. - A negative timeout (<0) on a job template is a true "no timeout" on the job. - A timeout of 0 on a job template defaults the job to the global timeout (which is no timeout by default). - A positive timeout sets the timeout for that job template.	Yes
Show changes	Enables you to see the changes made by Ansible tasks.	Yes
Instance groups	Choose Instance and Container Groups to associate with this job template. If the list is extensive, use the 🔍 icon to narrow the options. Job template instance groups contribute to the job scheduling criteria, see Job Runtime Behavior and Control where a job runs for rules. A System Administrator must grant you or your team permissions to be able to use an instance group in a job template. Use of a container group requires admin rights.	- Yes. If selected, you are providing the jobs preferred instance groups in order of preference. If the first group is out of capacity, later groups in the list are considered until one with capacity is available, at which point that is selected to run the job. - If you prompt for an instance group, what you enter replaces the normal instance group hierarchy and overrides all of the organizations' and inventories' instance groups. - The Instance Groups prompt shows up as its own step in a later prompt window.

Field	Options	Prompt on Launch
Job tags	Type and select the **Create** menu to specify which parts of the playbook should be executed. For more information and examples see Tags in the Ansible documentation.	Yes
Skip tags	Type and select the **Create** menu to specify certain tasks or parts of the playbook to skip. For more information and examples see Tags in the Ansible documentation.	Yes
Extra variables	● Pass extra command line variables to the playbook. This is the "-e" or "-extra-vars" command line parameter for ansible-playbook that is documented in the Ansible documentation at Defining variables at runtime. ● Give key or value pairs by using either YAML or JSON. These variables have a maximum value of precedence and overrides other variables specified elsewhere. The following is an example value: **git_branch: production release_version: 1.5**	Yes. If you want to be able to specify **extra_vars** on a schedule, you must select **Prompt on launch** for Variables on the job template, or enable a survey on the job template. Those answered survey questions become **extra_vars**.

4. You can set the following options for launching this template, if necessary:

   - **Privilege escalation**: If checked, you enable this playbook to run as an administrator. This is the equal of passing the **--become** option to the **ansible-playbook** command.

   - **Provisioning callback**: If checked, you enable a host to call back to automation controller through the REST API and start a job from this job template. For more information, see Provisioning Callbacks.

   - **Enable webhook**: If checked, you turn on the ability to interface with a predefined SCM system web service that is used to launch a job template. GitHub and GitLab are the supported SCM systems.

     - If you enable webhooks, other fields display, prompting for additional information:

     - **Webhook service**: Select which service to listen for webhooks from.

  - **Webhook URL**: Automatically populated with the URL for the webhook service to POST requests to.
  - **Webhook key**: Generated shared secret to be used by the webhook service to sign payloads sent to automation controller. You must configure this in the settings on the webhook service in order for automation controller to accept webhooks from this service.
  - **Webhook credential**: Optionally, give a GitHub or GitLab personal access token (PAT) as a credential to use to send status updates back to the webhook service.
    Before you can select it, the credential must exist.

    See Credential types to create one.
  - For additional information about setting up webhooks, see Working with Webhooks.
- **Concurrent jobs**: If checked, you are allowing jobs in the queue to run simultaneously if not dependent on one another. Check this box if you want to run job slices simultaneously. For more information, see Automation controller capacity determination and job impact.
- **Enable fact storage**: If checked, automation controller stores gathered facts for all hosts in an inventory related to the job running.
- **Prevent instance group fallback**: Check this option to allow only the instance groups listed in the **Instance Groups** field to run the job. If clear, all available instances in the execution pool are used based on the hierarchy described in Control where a job runs.

5. Click **Create job template**, when you have completed configuring the details of the job template.

Creating the template does not exit the job template page but advances to the Job Template **Details** tab. After saving the template, you can click **Launch template** to start the job. You can also click **Edit** to add or change the attributes of the template, such as permissions, notifications, view completed jobs, and add a survey (if the job type is not a scan). You must first save the template before launching, otherwise, **Launch template** remains disabled.

Verification

1. From the navigation panel, select **Automation Execution → Templates**.
2. Verify that the newly created template appears on the **Templates** page.

6.2. ADDING PERMISSIONS TO TEMPLATES

Use the following steps to add permissions for the team.

Procedure

1. From the navigation panel, select **Automation Execution → Templates**.
2. Select a template, and in the **Team Access** or **User Access** tab, click **Add roles**.
3. Select **Teams** or **Users** and click **Next**.
   - Select one or more users or teams from the list by clicking the check boxes next to the names to add them as members and click **Next**.

4. Choose the roles that you want users or teams to have. Ensure that you scroll down for a complete list of roles. Each resource has different options available.

5. Click **Finish** to apply the roles to the selected users or teams and to add them as members.

The window to add users and teams closes to display the updated roles assigned for each user and team

To remove roles for a particular user, click the ✕ icon next to its resource.

This launches a confirmation dialog, asking you to confirm the disassociation.

6.3. DELETING A JOB TEMPLATE

Before deleting a job template, ensure that it is not used in a workflow job template.

Procedure

1. Delete a job template by using one of these methods:
   - Select the checkbox next to one or more job templates. Click ⋮ and select **Delete template**.
   - Select the required job template, on the **Details** page click ⋮ and select **Delete template**.

> NOTE
>
> If deleting items that are used by other work items, a message opens listing the items that are affected by the deletion and prompts you to confirm the deletion. Some screens contain items that are invalid or previously deleted, and will fail to run. The following is an example of that message:

6.4. WORK WITH NOTIFICATIONS

From the navigation panel, select **Automation Execution → Administration → Notifiers**. You can review any notification integrations you have set up and their statuses, if they have run.

Use the toggles to enable or disable the notifications to use with your particular template. For more information, see Enable and disable notifications .

If no notifications have been set up, click **Add notifier** to create a new notification. For more information about configuring various notification types and extended messaging, see Notification types.

6.5. VIEW COMPLETED JOBS

The **Jobs** tab provides the list of job templates that have run. Click the expand icon next to each job to view the following details:

- Status
- ID and name
- Type of job
- Time started and completed

- Who started the job and which template, inventory, project, and credential were used.

You can filter the list of completed jobs using any of these criteria.

Sliced jobs that display on this list are labeled accordingly, with the number of sliced jobs that have run.

6.6. SCHEDULING JOB TEMPLATES

Access the schedules for a particular job template from the **Schedules** tab.

Procedure

- To schedule a job template, select the **Schedules** tab from the job template, and select the appropriate method:
  - If schedules are already set up, review, edit, enable or disable your schedule preferences.
  - If schedules have not been set up, see Schedules for more information.

If you select **Prompt on Launch** for the **Credentials field**, and you create or edit scheduling information for your job template, a **Prompt** option displays on the Schedules form.

You cannot remove the default machine credential in the **Prompt** dialog without replacing it with another machine credential before you can save it.

NOTE

To set **extra_vars** on schedules, you must select **Prompt on Launch** for **Variables** on the job template, or configure and enable a survey on the job template.

The answered survey questions then become **extra_vars**.

6.7. SURVEYS IN JOB TEMPLATES

Job types of **Run** or **Check** provide a way to set up surveys in the **Job Template** creation or editing screens. Surveys set extra variables for the playbook similar to **Prompt for Extra Variables** does, but in a user-friendly question and answer way. Surveys also permit for validation of user input. Select the **Survey** tab to create a survey.

Example

You can use surveys for several situations. For example, operations want to give developers a "push to stage" button that they can run without advance knowledge of Ansible. When launched, this task could prompt for answers to questions such as "What tag should we release?".

You can ask many types of questions, including multiple-choice questions.

6.7.1. Creating a survey

Procedure

1. From the navigation panel, select **Automation Execution → Templates**.
2. Select the job template you want to create a survey for.

3. From the **Survey** tab, click **Create survey question**.

4. A survey can consist of any number of questions. For each question, enter the following information:

   - **Question**: The question to ask the user.

   - Optional: **Description**: A description of what is being asked of the user.

   - **Answer variable name**: The Ansible variable name to store the user's response in. This is the variable to be used by the playbook. Variable names cannot contain spaces.

   - **Answer type**: Choose from the following question types:

     - **Text**: A single line of text. You can set the minimum and maximum length (in characters) for this answer.

     - **Textarea**: A multi-line text field. You can set the minimum and maximum length (in characters) for this answer.

     - **Password**: Responses are treated as sensitive information, much like an actual password is treated. You can set the minimum and maximum length (in characters) for this answer.

     - **Multiple Choice (single select)**: A list of options, of which only one can be selected at a time. Enter the options, one per line, in the **Multiple Choice Options** field.

     - **Multiple Choice (multiple select)**: A list of options, any number of which can be selected at a time. Enter the options, one per line, in the **Multiple Choice Options** field.

     - **Integer**: An integer number. You can set the minimum and maximum length (in characters) for this answer.

     - **Float**: A decimal number. You can set the minimum and maximum length (in characters) for this answer.

   - **Required**: Whether or not an answer to this question is required from the user.

   - **Minimum length** and **Maximum length**: Specify if a certain length in the answer is required.

   - **Default answer**: The default answer to the question. This value is pre-filled in the interface and is used if the answer is not provided by the user.

5. Once you have entered the question information, click **Create question** to add the question.
   The survey question displays in the **Survey** list. For any question, you can click ✎ to edit it.

   Check the box next to each question and click **Delete** to delete the question, or use the toggle option in the menu bar to enable or disable the survey prompts.

   If you have more than one survey question, click **Edit Order** to rearrange the order of the questions by clicking and dragging on the grid icon.

6. To add more questions, click **Add**.

6.7.2. Optional survey questions

The **Required** setting on a survey question determines whether the answer is optional or not for the user interacting with it.

Optional survey variables can also be passed to the playbook in **extra_vars**.

- If a non-text variable (input type) is marked as optional, and is not filled in, no survey **extra_var** is passed to the playbook.
- If a text input or text area input is marked as optional, is not filled in, and has a minimum **length > 0**, no survey **extra_var** is passed to the playbook.
- If a text input or text area input is marked as optional, is not filled in, and has a minimum **length === 0**, that survey **extra_var** is passed to the playbook, with the value set to an empty string ("").

6.8. LAUNCHING A JOB TEMPLATE

A benefit of automation controller is the push-button deployment of Ansible playbooks. You can configure a template to store all the parameters that you would normally pass to the Ansible Playbook on the command line. In addition to the playbooks, the template passes the inventory, credentials, extra variables, and all options and settings that you can specify on the command line.

Easier deployments drive consistency, by running your playbooks the same way each time, and allowing you to delegate responsibilities.

Procedure

- Launch a job template by using one of these methods:
  - From the navigation panel, select **Automation Execution → Templates** and click **Launch template** 🚀 next to the job template.
  - In the job template **Details** tab of the job template you want to launch, click **Launch template**.

A job can require additional information to run. The following data can be requested at launch:

- Credentials that were setup
- The option **Prompt on Launch** is selected for any parameter
- Passwords or passphrases that have been set to **Ask**
- A survey, if one has been configured for the job templates
- Extra variables, if requested by the job template

NOTE

If a job has user-provided values, then those are respected upon relaunch. If the user did not specify a value, then the job uses the default value from the job template. Jobs are not relaunched as-is. They are relaunched with the user prompts re-applied to the job template.

If you give values on one tab, return to a previous tab, continuing to the next tab results in having to re-provide values on the rest of the tabs. Ensure that you complete the tabs in the order that the prompts appear.

When launching, automation controller automatically redirects the web browser to the **Job Status** page for this job under the **Jobs** tab.

You can re-launch the most recent job from the list view to re-run on all hosts or just failed hosts in the specified inventory. For more information, see the Jobs in automation controller section.

When slice jobs are running, job lists display the workflow and job slices, and a link to view their details individually.

NOTE

You can launch jobs in bulk by using the newly added endpoint in the API, **/api/v2/bulk/job_launch**. This endpoint accepts JSON and you can specify a list of unified job templates (such as job templates and project updates) to launch. The user must have the appropriate permission to launch all the jobs. If all jobs are not launched an error is returned indicating why the operation was not able to complete. Use the **OPTIONS** request to return relevant schema. For more information, see the Bulk endpoint of the Reference section of the Automation Controller API Guide.

6.9. COPYING A JOB TEMPLATE

If you copy a job template, it does not copy any associated schedule, notifications, or permissions. Schedules and notifications must be recreated by the user or administrator creating the copy of the job template. The user copying the Job Template is granted administrator permission, but no permissions are assigned (copied) to the job template.

Procedure

1. From the navigation panel, select **Automation Execution → Templates**.
2. Click ⋮ and the copy ⧉ icon associated with the template that you want to copy.
   - The new template with the name of the template from which you copied and a timestamp displays in the list of templates.
3. Click to open the new template and click **Edit template**.
4. Replace the contents of the **Name** field with a new name, and give or change the entries in the other fields to complete this page.
5. Click **Save job template**.

6.10. SCAN JOB TEMPLATES

Scan jobs are no longer supported starting with automation controller 3.2. This system tracking feature was used as a way to capture and store facts as historical data. Facts are now stored in the controller through fact caching. For more information, see Fact Caching.

Job template scan jobs in your system before automation controller 3.2, are converted to type run, like normal job templates. They retain their associated resources, such as inventories and credentials. By default, job template scan jobs that do not have a related project are assigned a special playbook. You can also specify a project with your own scan playbook. A project is created for each organization that points to awx-facts-playbooks and the job template was set to the playbook: https://github.com/ansible/tower-fact-modules/blob/master/scan_facts.yml.

6.10.1. Fact scan playbooks

The scan job playbook, **scan_facts.yml**, contains invocations of three **fact scan modules** - packages, services, and files, along with Ansible's standard fact gathering. The **scan_facts.yml** playbook file is similar to this:

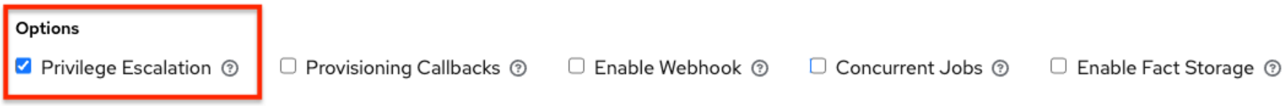
```
- hosts: all
  vars:
    scan_use_checksum: false
    scan_use_recursive: false
  tasks:
    - scan_packages:
    - scan_services:
    - scan_files:
        paths: '{{ scan_file_paths }}'
        get_checksum: '{{ scan_use_checksum }}'
        recursive: '{{ scan_use_recursive }}'
      when: scan_file_paths is defined
```

The **scan_files** fact module is the only module that accepts parameters, passed through **extra_vars** on the scan job template:

scan_file_paths: **/tmp/scan_use_checksum**: true **scan_use_recursive**: true

- The **scan_file_paths** parameter can have multiple settings (such as **/tmp/** or **/var/log**).
- The **scan_use_checksum** and **scan_use_recursive** parameters can also be set to false or omitted. An omission is the same as a false setting.

Scan job templates should enable **become** and use **credentials** for which **become** is a possibility. You can enable **become** by checking **Privilege Escalation** from the options list:



Options

☑ Privilege Escalation ⓘ □ Provisioning Callbacks ⓘ □ Enable Webhook ⓘ □ Concurrent Jobs ⓘ □ Enable Fact Storage ⓘ

6.10.2. Supported OSes for scan_facts.yml

If you use the **scan_facts.yml** playbook with use fact cache, ensure that you are using one of the following supported operating systems:

- Red Hat Enterprise Linux 5, 6, 7, 8, and 9
- Ubuntu 23.04 (Support for Ubuntu is deprecated and will be removed in a future release)
- OEL 6 and 7
- SLES 11 and 12
- Debian 6, 7, 8, 9, 10, 11, and 12
- Fedora 22, 23, and 24

- Amazon Linux 2023.1.20230912

Some of these operating systems require initial configuration to run python or have access to the python packages, such as **python-apt**, which the scan modules depend on.

6.10.3. Pre-scan setup

The following are examples of playbooks that configure certain distributions so that scan jobs can be run against them:

```
Bootstrap Ubuntu (16.04)
---
- name: Get Ubuntu 16, and on ready
 hosts: all
 sudo: yes
 gather_facts: no
 tasks:
 - name: install python-simplejson
   raw: sudo apt-get -y update
   raw: sudo apt-get -y install python-simplejson
   raw: sudo apt-get install python-apt

Bootstrap Fedora (23, 24)
---
- name: Get Fedora ready
 hosts: all
 sudo: yes
 gather_facts: no
 tasks:
 - name: install python-simplejson
   raw: sudo dnf -y update
   raw: sudo dnf -y install python-simplejson
   raw: sudo dnf -y install rpm-python
```

6.10.4. Custom fact scans

A playbook for a custom fact scan is similar to the example in the Fact scan playbooks section. For example, a playbook that only uses a custom **scan_foo** Ansible fact module looks similar to this:

```
scan_foo.py:
def main():
    module = AnsibleModule(
        argument_spec = dict())


    foo = [
      {
        "hello": "world"
      },
      {
        "foo": "bar"
      }
    ]
    results = dict(ansible_facts=dict(foo=foo))
    module.exit_json(**results)
```

```

main()
```

To use a custom fact module, ensure that it lives in the **/library/** subdirectory of the Ansible project used in the scan job template. This fact scan module returns a hard-coded set of facts:

```
[
  {
    "hello": "world"
  },
  {
    "foo": "bar"
  }
 ]
```

For more information, see the Developing modules section of the Ansible documentation.

6.10.5. Fact caching

Automation controller can store and retrieve facts on a per-host basis through an Ansible Fact Cache plugin. This behavior is configurable on a per-job template basis. Fact caching is turned off by default but can be enabled to serve fact requests for all hosts in an inventory related to the job running. This enables you to use job templates with **--limit** while still having access to the entire inventory of host facts. You can specify a global timeout setting that the plugin enforces per-host, (in seconds) from the navigation panel, select **Settings** → **Job** and edit the **Per-Host Ansible Fact Cache Timeout**field.

After launching a job that uses fact cache (**use_fact_cache=True**), each host's **ansible_facts** are all stored by the controller in the job's inventory.

The Ansible Fact Cache plugin that includes automation controller is enabled on jobs with fact cache enabled (**use_fact_cache=True**).

When a job that has fact cache enabled (**use_fact_cache=True**) has run, automation controller restores all records for the hosts in the inventory. Any records with update times newer than the currently stored facts per-host are updated in the database.

New and changed facts are logged through automation controller's logging facility. Specifically, to the **system_tracking namespace** or logger. The logging payload includes the following fields:

- **host_name**
- **inventory_id**
- **ansible_facts**

ansible facts is a dictionary of all Ansible facts for **host_name** in the automation controller inventory, **inventory_id**.

NOTE

If a hostname includes a forward slash (/), fact cache does not work for that host. If you have an inventory with 100 hosts and one host has a / in the name, the remaining 99 hosts still collect facts.

6.10.6. Benefits of fact caching

Fact caching saves you time over running fact gathering. If you have a playbook in a job that runs against a thousand hosts and forks, you can spend 10 minutes gathering facts across all of those hosts. However, if you run a job on a regular basis, the first run of it caches these facts and the next run pulls them from the database. This reduces the runtime of jobs against large inventories, including Smart Inventories.

> NOTE
>
> Do not change the ansible.cfg file to apply fact caching. Custom fact caching could conflict with the controller's fact caching feature. You must use the fact caching module that includes automation controller.

You can select to use cached facts in your job by checking the **Enable fact storage** option when you create or edit a job template.

To clear facts, run the Ansible **clear_facts** meta task. The following is an example playbook that uses the Ansible **clear_facts** meta task.

```
- hosts: all
  gather_facts: false
  tasks:
    - name: Clear gathered facts from all currently targeted hosts
      meta: clear_facts
```

You can find the API endpoint for fact caching at:

http://<controller server name>/api/v2/hosts/x/ansible_facts

6.11. USE CLOUD CREDENTIALS WITH A CLOUD INVENTORY

Cloud Credentials can be used when syncing a cloud inventory. They can also be associated with a job template and included in the runtime environment for use by a playbook. The following Cloud Credentials are supported:

- Openstack
- Amazon Web Services
- Google
- Azure
- VMware

6.11.1. OpenStack

The following sample playbook invokes the **nova_compute** Ansible OpenStack cloud module and requires credentials:

- **auth_url**
- **username**

- **password**
- **project name**

These fields are made available to the playbook through the environmental variable **OS_CLIENT_CONFIG_FILE**, which points to a YAML file written by the controller based on the contents of the cloud credential. The following sample playbooks load the YAML file into the Ansible variable space:

- OS_CLIENT_CONFIG_FILE example:

```
clouds:
  devstack:
    auth:
      auth_url: http://devstack.yoursite.com:5000/v2.0/
      username: admin
      password: your_password_here
      project_name: demo
```

- Playbook example:

```
- hosts: all
  gather_facts: false
  vars:
    config_file: "{{ lookup('env', 'OS_CLIENT_CONFIG_FILE') }}"
    nova_tenant_name: demo
    nova_image_name: "cirros-0.3.2-x86_64-uec"
    nova_instance_name: autobot
    nova_instance_state: 'present'
    nova_flavor_name: m1.nano


    nova_group:
      group_name: antarctica
      instance_name: deceptacon
      instance_count: 3
  tasks:
    - debug: msg="{{ config_file }}"
    - stat: path="{{ config_file }}"
      register: st
    - include_vars: "{{ config_file }}"
      when: st.stat.exists and st.stat.isreg


    - name: "Print out clouds variable"
      debug: msg="{{ clouds|default('No clouds found') }}"


    - name: "Setting nova instance state to: {{ nova_instance_state }}"
      local_action:
        module: nova_compute
        login_username: "{{ clouds.devstack.auth.username }}"
        login_password: "{{ clouds.devstack.auth.password }}"
```

6.11.2. Amazon Web Services

Amazon Web Services (AWS) cloud credentials are exposed as the following environment variables during playbook execution (in the job template, choose the cloud credential needed for your setup):

- **AWS_ACCESS_KEY_ID**
- **AWS-SECRET_ACCESS_KEY**

Each AWS module implicitly uses these credentials when run through the controller without having to set the **aws_access_key_id** or **aws_secret_access_key** module options.

6.11.3. Google

Google cloud credentials are exposed as the following environment variables during playbook execution (in the job template, choose the cloud credential needed for your setup):

- **GCE_EMAIL**
- **GCE_PROJECT**
- **GCE_CREDENTIALS_FILE_PATH**

Each Google module implicitly uses these credentials when run through the controller without having to set the **service_account_email**, **project_id**, or **pem_file** module options.

6.11.4. Azure

Azure cloud credentials are exposed as the following environment variables during playbook execution (in the job template, choose the cloud credential needed for your setup):

- **AZURE_SUBSCRIPTION_ID**
- **AZURE_CERT_PATH**

Each Azure module implicitly uses these credentials when run via the controller without having to set the **subscription_id** or **management_cert_path** module options.

6.11.5. VMware

VMware cloud credentials are exposed as the following environment variables during playbook execution (in the job template, choose the cloud credential needed for your setup):

- **VMWARE_USER**
- **VMWARE_PASSWORD**
- **VMWARE_HOST**

The following sample playbook demonstrates the usage of these credentials:

```
- vsphere_guest:
    vcenter_hostname: "{{ lookup('env', 'VMWARE_HOST') }}"
    username: "{{ lookup('env', 'VMWARE_USER') }}"
    password: "{{ lookup('env', 'VMWARE_PASSWORD') }}"
    guest: newvm001
    from_template: yes
    template_src: linuxTemplate
```

```
cluster: MainCluster
resource_pool: "/Resources"
vm_extra_config:
  folder: MyFolder
```

6.12. PROVISIONING CALLBACKS

Provisioning Callbacks are a feature of automation controller that enable a host to start a playbook run against itself, rather than waiting for a user to launch a job to manage the host from the automation controller console.

Provisioning Callbacks are only used to run playbooks on the calling host and are meant for cloud bursting. Cloud bursting is a cloud computing configuration that enables a private cloud to access public cloud resources by "bursting" into a public cloud when computing demand spikes.

Example

New instances with a need for client to server communication for configuration, such as transmitting an authorization key, not to run a job against another host. This provides for automatically configuring the following:

- A system after it has been provisioned by another system (such as AWS auto-scaling, or an OS provisioning system like kickstart or preseed).
- Launching a job programmatically without invoking the automation controller API directly.

The job template launched only runs against the host requesting the provisioning.

This is often accessed with a firstboot type script or from **cron**.

6.12.1. Enabling Provisioning Callbacks

Procedure

- To enable callbacks, check the **Provisioning callback** option in the job template. This displays **Provisioning callback details** for the job template.

> **NOTE**
>
> If you intend to use automation controller’s provisioning callback feature with a dynamic inventory, set **Update on Launch** for the inventory group used in the job template.

Callbacks also require a host config key, to ensure that foreign hosts with the URL cannot request configuration. Give a custom value for the **Host config key**. The host key can be reused across many hosts to apply this job template against multiple hosts. If you want to control what hosts are able to request configuration, you can change the key can at any time.

To callback manually using REST:

Procedure

1. Examine the callback URL in the UI, in the form: https://<CONTROLLER_SERVER_NAME>/api/v2/job_templates/7/callback/

- The "7" in the sample URL is the job template ID in automation controller.

2. Ensure that the request from the host is a POST. The following is an example using **curl** (all on a single line):

```
curl -k -f -i -H 'Content-Type:application/json' -XPOST -d '{"host_config_key": "redhat"}' \
                  https://<CONTROLLER_SERVER_NAME>/api/v2/job_templates/7/callback/
```

3. Ensure that the requesting host is defined in your inventory for the callback to succeed.

Troubleshooting

If automation controller fails to locate the host either by name or IP address in one of your defined inventories, the request is denied. When running a job template in this way, ensure that the host initiating the playbook run against itself is in the inventory. If the host is missing from the inventory, the job template fails with a **No Hosts Matched** type error message.

If your host is not in the inventory and **Update on Launch** is checked for the inventory group, automation controller attempts to update cloud based inventory sources before running the callback.

Verification

Successful requests result in an entry on the **Jobs** tab, where you can view the results and history. You can access the callback by using REST, but the suggested method of using the callback is to use one of the example scripts that includes automation controller:

- **/usr/share/awx/request_tower_configuration.sh** (Linux/UNIX)
- **/usr/share/awx/request_tower_configuration.ps1** (Windows)

Their usage is described in the source code of the file by passing the **-h** flag, as the following shows:

```
./request_tower_configuration.sh -h
Usage: ./request_tower_configuration.sh <options>


Request server configuration from Ansible Tower.


OPTIONS:
 -h      Show this message
 -s      Controller server (e.g. https://ac.example.com) (required)
 -k      Allow insecure SSL connections and transfers
 -c      Host config key (required)
 -t      Job template ID (required)
 -e      Extra variables
```

This script can retry commands and is therefore a more robust way to use callbacks than a simple **curl** request. The script retries once per minute for up to ten minutes.

NOTE

This is an example script. Edit this script if you need more dynamic behavior when detecting failure scenarios, as any non-200 error code may not be a transient error requiring retry.

You can use callbacks with dynamic inventory in automation controller. For example, when pulling cloud inventory from one of the supported cloud providers. In these cases, along with setting **Update On Launch**, ensure that you configure an inventory cache timeout for the inventory source, to avoid hammering of your cloud's API endpoints. Since the **request_tower_configuration.sh** script polls once per minute for up to ten minutes, a suggested cache invalidation time for inventory (configured on the inventory source itself) would be one or two minutes.

Running the **request_tower_configuration.sh** script from a cron job is not recommended, however, a suggested cron interval is every 30 minutes. Repeated configuration can be handled by scheduling automation controller so that the primary use of callbacks by most users is to enable a base image that is bootstrapped into the latest configuration when coming online. Running at first boot is best practice. First boot scripts are init scripts that typically self-delete, so you set up an init script that calls a copy of the **request_tower_configuration.sh** script and make that into an auto scaling image.

6.12.2. Passing extra variables to Provisioning Callbacks

You can pass **extra_vars** in Provisioning Callbacks the same way you can in a regular job template. To pass **extra_vars**, the data sent must be part of the body of the POST as application or JSON, as the content type.

Procedure

- Pass extra variables by using one of these methods:
  - Use the following JSON format as an example when adding your own **extra_vars** to be passed:

    ```
    '{"extra_vars": {"variable1":"value1","variable2":"value2",...}}'
    ```

  - Pass extra variables to the job template call using **curl**:

    ```
    root@localhost:~$ curl -f -H 'Content-Type: application/json' -XPOST \
    -d '{"host_config_key": "redhat", "extra_vars": "{\"foo\": \"bar\"}"}' \
    https://<CONTROLLER_SERVER_NAME>/api/v2/job_templates/7/callback
    ```

For more information, see Launching Jobs with Curl in *Configuring automation execution*.

6.13. EXTRA VARIABLES

When you pass survey variables, they are passed as extra variables (**extra_vars**) within automation controller. However, passing extra variables to a job template (as you would do with a survey) can override other variables being passed from the inventory and project.

By default, **extra_vars** are marked as **!unsafe** unless you specify them on the Job Template's Extra Variables section. These are trusted, because they can only be added by users with enough privileges to add or edit a Job Template. For example, nested variables do not expand when entered as a prompt, as the Jinja brackets are treated as a string. For more information about unsafe variables, see Unsafe or raw strings.

NOTE

extra_vars passed to the job launch API are only honored if one of the following is true:

- They correspond to variables in an enabled survey.
- **ask_variables_on_launch** is set to **True**.

Example

You have a defined variable for an inventory for **debug = true**. It is possible that this variable, **debug = true**, can be overridden in a job template survey.

To ensure the variables that you pass are not overridden, ensure they are included by redefining them in the survey. You can define extra variables at the inventory, group, and host levels.

If you are specifying the **ALLOW_JINJA_IN_EXTRA_VARS** parameter, see the The ALLOW_JINJA_IN_EXTRA_VARS variable section of *Configuring automation execution* to configure it.

The job template extra variables dictionary is merged with the survey variables.

The following are some simplified examples of **extra_vars** in YAML and JSON formats:

- The configuration in YAML format:

```
launch_to_orbit: true
satellites:
  - sputnik
  - explorer
  - satcom
```

- The configuration in JSON format:

```
{
  "launch_to_orbit": true,
  "satellites": ["sputnik", "explorer", "satcom"]
}
```

The following table notes the behavior (hierarchy) of variable precedence in automation controller as it compares to variable precedence in Ansible.

Table 6.1. Automation controller Variable Precedence Hierarchy (last listed wins)

Ansible	automation controller
role defaults	role defaults
dynamic inventory variables	dynamic inventory variables
inventory variables	automation controller inventory variables
inventory **group_vars**	automation controller group variables

Ansible	automation controller
inventory **host_vars**	automation controller host variables
playbook **group_vars**	playbook **group_vars**
playbook **host_vars**	playbook **host_vars**
host facts	host facts
registered variables	registered variables
set facts	set facts
play variables	play variables
play **vars_prompt**	(not supported)
play **vars_files**	play **vars_files**
role and include variables	role and include variables
block variables	block variables
task variables	task variables
extra variables	Job Template extra variables
	Job Template Survey (defaults)
	Job Launch extra variables

6.13.1. Relaunch a job template

Instead of manually relaunching a job, a relaunch is denoted by setting **launch_type** to **relaunch**. The relaunch behavior deviates from the launch behavior in that it does not inherit **extra_vars**.

Job relaunching does not go through the inherit logic. It uses the same **extra_vars** that were calculated for the job being relaunched.

Example

You launch a job template with no **extra_vars** which results in the creation of a job called **j1**. Then you edit the job template and add **extra_vars** (such as adding **"{ "hello": "world" }"**).

Relaunching **j1** results in the creation of **j2**, but because there is no inherit logic and **j1** has no extra_vars, **j2** does not have any **extra_vars**.

If you launch the job template with the **extra_vars** that you added after the creation of j1, the relaunch job created (j3) includes the extra_vars. Relaunching j3 results in the creation of j4, which also includes **extra_vars**.

CHAPTER 7. JOB SLICING

A sliced job refers to the concept of a distributed job. Use distributed jobs for running a job across a large number of hosts. You can then run many ansible-playbooks, each on a subset of an inventory that you can be schedule in parallel across a cluster.

By default, Ansible runs jobs from a single control instance. For jobs that do not require cross-host orchestration, job slicing takes advantage of automation controller's ability to distribute work to many nodes in a cluster.

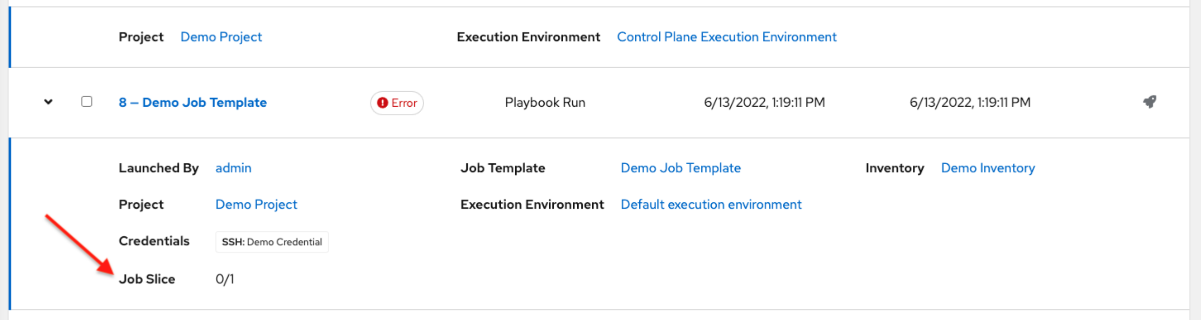
Job slicing works by adding a Job Template field **job_slice_count**, which specifies the number of jobs into which to slice the Ansible run. When this number is greater than **1**, automation controller generates a workflow from a job template instead of a job. The inventory is distributed evenly among the slice jobs. The workflow job is then started, and proceeds as though it were a normal workflow.

When launching a job, the API returns either a job resource (if job_slice_count = 1) or a workflow job resource. The corresponding User Interface (UI) redirects to the appropriate screen to display the status of the run.

7.1. JOB SLICE CONSIDERATIONS

When setting up job slices, consider the following:

- A sliced job creates a workflow job, which then creates jobs.
- A job slice consists of a job template, an inventory, and a slice count.
- When executed, a sliced job splits each inventory into several "slice size" chunks. It then queues jobs of ansible-playbook runs on each chunk of the appropriate inventory. The inventory fed into ansible-playbook is a shortened version of the original inventory that only has the hosts in that particular slice. The completed sliced job that displays on the **Jobs** list are labeled accordingly, with the number of sliced jobs that have run:



Project Demo Project
Execution Environment Control Plane Execution Environment

8 – Demo Job Template | Error | Playbook Run | 6/13/2022, 1:19:11 PM | 6/13/2022, 1:19:11 PM

Launched By admin
Job Template Demo Job Template
Inventory Demo Inventory
Project Demo Project
Execution Environment Default execution environment
Credentials SSH: Demo Credential
Job Slice 0/1

- These sliced jobs follow normal scheduling behavior (number of forks, queuing due to capacity, assignation to instance groups based on inventory mapping).

NOTE

Job slicing is intended to scale job executions horizontally. Enabling job slicing on a job template divides an inventory to be acted upon in the number of slices configured at launch time and then starts a job for each slice.

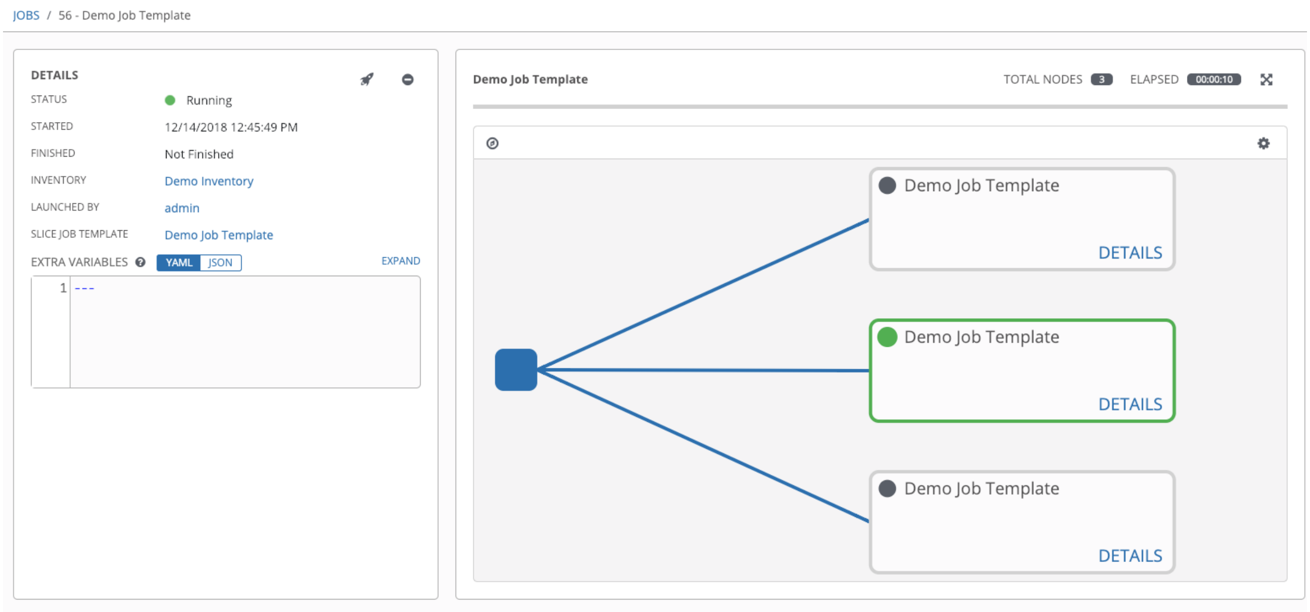
Normally, the number of slices is equal to or less than the number of automation controller nodes. You can set an extremely high number of job slices but it can cause performance degradation. The job scheduler is not designed to simultaneously schedule thousands of workflow nodes, which are what the sliced jobs become.

- Sliced job templates with prompts or extra variables behave the same as standard job templates, applying all variables and limits to the entire set of slice jobs in the resulting workflow job. However, when passing a limit to a sliced job, if the limit causes slices to have no hosts assigned, those slices will fail, causing the overall job to fail.
- A job slice job status of a distributed job is calculated in the same manner as workflow jobs. It fails if there are any unhandled failures in its sub-jobs.

- Any job that intends to orchestrate across hosts (rather than just applying changes to individual hosts) must not be configured as a slice job.
- Any job that does, can fail, and automation controller does not try to discover or account for playbooks that fail when run as slice jobs.

7.2. JOB SLICE EXECUTION BEHAVIOR

When jobs are sliced, they can run on any node. Insufficient capacity in the system can cause some to run at a different time. When slice jobs are running, job details display the workflow and job slices currently running, and a link to view their details individually.


JOBS / 56 - Demo Job Template

DETAILS

STATUS: Running
STARTED: 12/14/2018 12:45:49 PM
FINISHED: Not Finished
INVENTORY: Demo Inventory
LAUNCHED BY: admin
SLICE JOB TEMPLATE: Demo Job Template
EXTRA VARIABLES: YAML

```
---
```

Demo Job Template

TOTAL NODES 3 ELAPSED 00:00:10

By default, job templates are not normally configured to execute simultaneously (you must check **allow_simultaneous** in the API or **Concurrent jobs** in the UI). Slicing overrides this behavior and implies **allow_simultaneous** even if that setting is clear. See Job templates for information about how to specify this, and the number of job slices on your job template configuration.

The Job templates section provides additional detail on performing the following operations in the UI:

- Launch workflow jobs with a job template that has a slice number greater than one.
- Cancel the whole workflow or individual jobs after launching a slice job template.
- Relaunch the whole workflow or individual jobs after slice jobs finish running.
- View the details about the workflow and slice jobs after launching a job template.
- Search slice jobs specifically after you create them, according to the next section, "Searching job slices").

7.3. SEARCHING JOB SLICES

To make it easier to find slice jobs, use the search functionality to apply a search filter to:

- Job lists to show only slice jobs
- Job lists to show only parent workflow jobs of job slices
- Job template lists to only show job templates that produce slice jobs

Procedure

- Search for slice jobs by using one of the following methods:
  - To show only slice jobs in job lists, as with most cases, you can filter either on the type (jobs here) or **unified_jobs**:

    ```
    /api/v2/jobs/?job_slice_count__gt=1
    ```

  - To show only parent workflow jobs of job slices:

    ```
    /api/v2/workflow_jobs/?job_template__isnull=false
    ```

  - To show only job templates that produce slice jobs:

    ```
    /api/v2/job_templates/?job_slice_count__gt=1
    ```

CHAPTER 8. WORKFLOW JOB TEMPLATES

You can create both Job templates and Workflow job templates from **Automation Execution → Templates**.

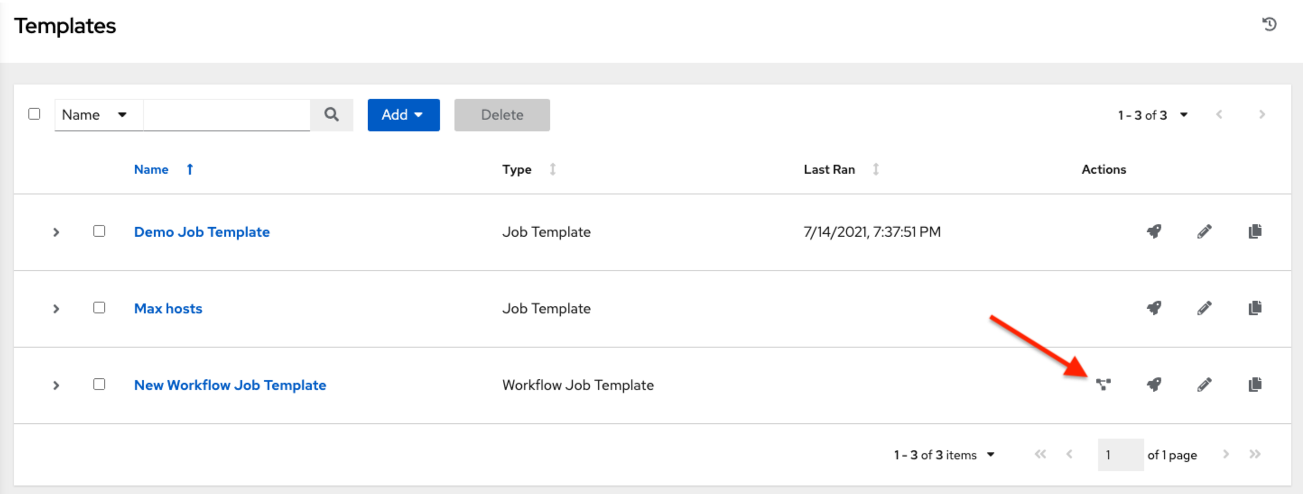
For Job templates, see Job templates.

A workflow job template links together a sequence of disparate resources that tracks the full set of jobs that were part of the release process as a single unit. These resources include the following:

- Job templates
- Workflow job templates
- Project syncs
- Inventory source syncs

The **Templates** page shows the workflow and job templates that are currently available. The default view is collapsed (Compact), showing the template name, template type, and the statuses of the jobs that have run by using that template. You can click the arrow next to each entry to expand and view more information. This list is sorted alphabetically by name, but you can sort by other criteria, or search by various fields and attributes of a template. From this screen you can launch 🚀, edit ✏, and copy ⧉ a workflow job template.

Only workflow templates have the workflow visualizer ⑂ icon as a shortcut for accessing the workflow editor.



Templates

☐ Name ▾ | Add ▾ | Delete | 1 - 3 of 3 ▾

		Name ↑	Type	Last Ran	Actions
›	☐	Demo Job Template	Job Template	7/14/2021, 7:37:51 PM	🚀 ✏ ⧉
›	☐	Max hosts	Job Template		🚀 ✏ ⧉
›	☐	New Workflow Job Template	Workflow Job Template		⑂ 🚀 ✏ ⧉

1 - 3 of 3 items ▾ | 1 | of 1 page

> **NOTE**
>
> Workflow templates can be used as building blocks for another workflow template. You can enable **Prompt on Launch** by setting up several settings in a workflow template, which you can edit at the workflow job template level. These do not affect the values assigned at the individual workflow template level. For further instructions, see the Workflow Visualizer section.

8.1. CREATING A WORKFLOW JOB TEMPLATE

To create a new workflow job template, complete the following steps:

IMPORTANT

If you set a limit to a workflow template, it is not passed down to the job template unless you check **Prompt on launch** for the limit. This can lead to playbook failures if the limit is mandatory for the playbook that you are running.

Procedure

1. From the navigation panel, select **Automation Execution → Templates**.
2. On the **Templates** list view, select **Create workflow job template** from the **Create template** list.
3. Enter the appropriate details in the following fields:

NOTE

If a field has the **Prompt on launch** checkbox selected, either launching the workflow template, or using the workflow template within another workflow template, you are prompted for the value for that field. Most prompted values override any values set in the job template. Exceptions are noted in the following table.

Field	Options	Prompt on Launch
Name	Enter a name for the job.	N/A
Description	Enter an arbitrary description as appropriate (optional).	N/A
Organization	Choose the organization to use with this template from the organizations available to the logged in user.	N/A
Inventory	Optionally, select the inventory to use with this template from the inventories available to the logged in user.	Yes

Field	Options	Prompt on Launch
Limit	A host pattern to further constrain the list of hosts managed or affected by the playbook. You can separate many patterns by colons (:). As with core Ansible: ● a:b means "in group a or b" ● a:b:&c means "in a or b but must be in c" ● a:!b means "in a, and definitely not in b" For more information see, Patterns: targeting hosts and groups in the Ansible documentation.	Yes If selected, even if a default value is supplied, you are prompted upon launch to select a limit.
Source control branch	Select a branch for the workflow. This branch is applied to all workflow job template nodes that prompt for a branch.	Yes

Field	Options	Prompt on Launch
Labels	• Optionally, supply labels that describe this workflow job template, such as **dev** or **test**. Use labels to group and filter workflow job templates and completed jobs in the display. • Labels are created when they are added to the workflow template. Labels are associated to a single Organization using the Project that is provided in the workflow template. Members of the Organization can create labels on a workflow template if they have edit permissions (such as the admin role). • Once you save the job template, the labels appear in the workflow job template **Details** view. • Labels are only applied to the workflow templates not the job template nodes that are used in the workflow. • Select ✖ beside a label to remove it. When a label is removed, it is no longer associated with that particular Job or Job Template, but it remains associated with any other jobs that reference it.	Yes If selected, even if a default value is supplied, you are prompted when launching to supply additional labels, if needed. - You cannot delete existing labels, selecting ✖ only removes the newly added labels, not existing default labels.
Job tags	Type and select the **Create** drop-down to specify which parts of the playbook should run. For more information and examples see Tags in the Ansible documentation.	Yes

Field	Options	Prompt on Launch
Skip tags	Type and select the **Create** drop-down to specify certain tasks or parts of the playbook to skip. For more information and examples see Tags in the Ansible documentation.	Yes
Extra variables	- Pass extra command line variables to the playbook. This is the "-e" or "-extra-vars" command line parameter for ansible-playbook that is documented in the Ansible documentation at Controlling how Ansible behaves: precedence rules. - Give key or value pairs by using either YAML or JSON. These variables have a maximum value of precedence and overrides other variables specified elsewhere. The following is an example value: **git_branch: production release_version: 1.5**	Yes If you want to be able to specify **extra_vars** on a schedule, you must select **Prompt on launch** for **Extra variables** on the workflow job template, or enable a survey on the job template. Those answered survey questions become **extra_vars**. For more information about extra variables, see Extra Variables.

4. Specify the following **Options** for launching this template, if necessary:

   - Check **Enable webhook** to turn on the ability to interface with a predefined SCM system web service that is used to launch a workflow job template. GitHub and GitLab are the supported SCM systems.

     - If you enable webhooks, other fields display, prompting for additional information:

       - **Webhook service**: Select which service to listen for webhooks from.

       - **Webhook URL**: Automatically populated with the URL for the webhook service to POST requests to.

       - **Webhook key**: Generated shared secret to be used by the webhook service to sign payloads sent to automation controller. You must configure this in the settings on the webhook service so that webhooks from this service are accepted in automation controller. For additional information about setting up webhooks, see Working with Webhooks.

   - Check **Enable concurrent jobs** to allow simultaneous runs of this workflow. For more information, see Automation controller capacity determination and job impact .

5. When you have completed configuring the workflow template, click **Create workflow job template**.

Saving the template exits the workflow template page and the workflow visualizer opens where you can build a workflow. For more information, see the Workflow visualizer section. Otherwise, select one of these methods:

- Close the workflow visualizer to return to the **Details** tab of the newly saved template. There you can complete the following tasks:
    - Review, edit, add permissions, notifications, schedules, and surveys
    - View completed jobs
    - Build a workflow template
- Click **Launch template** to start the workflow.

> NOTE
>
> Save the template before launching, or **Launch template** remains disabled. The **Notifications** tab is only present after you save the template.

8.2. WORK WITH PERMISSIONS

Click the **Team Access** or **User Access** tab to review, grand, edit, and remove associated permissions for users along with team members.

Click **Add roles** to create new permissions for this workflow template by following the prompts to assign them.

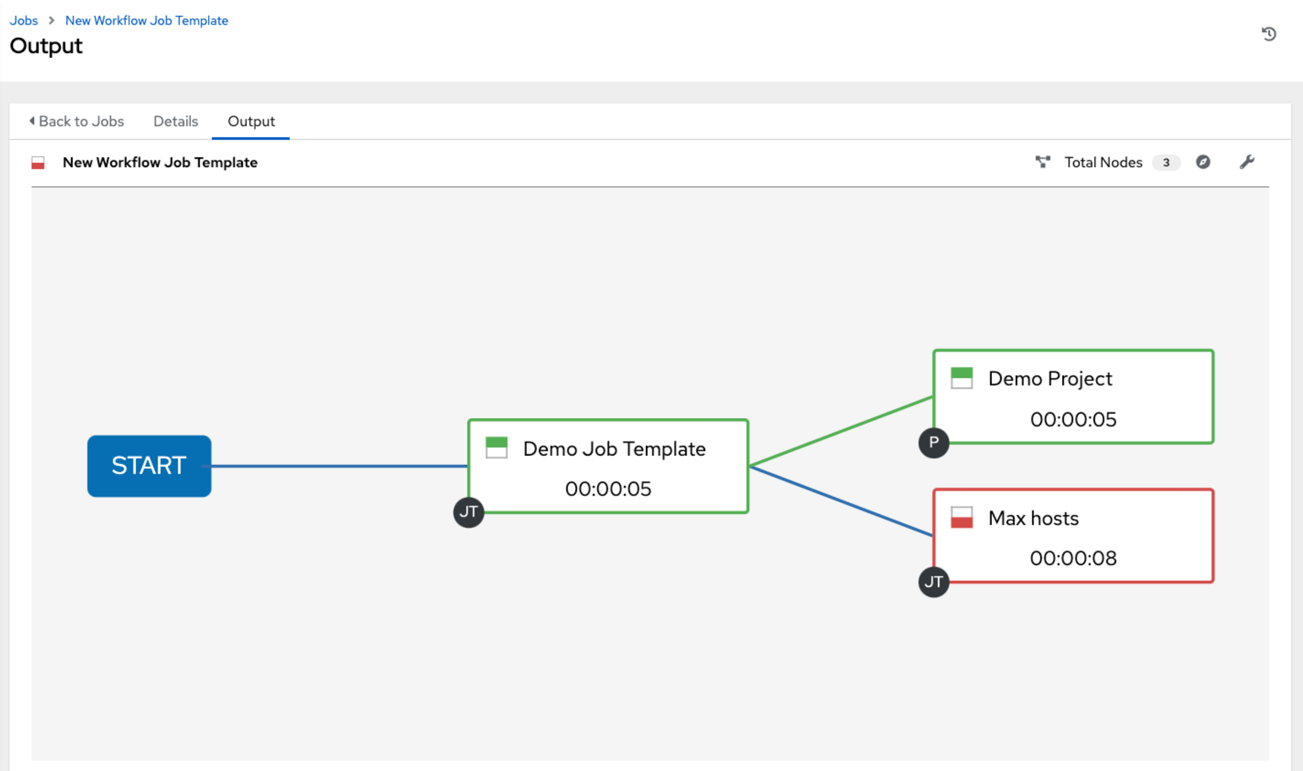
8.3. WORK WITH NOTIFICATIONS

For information on working with notifications in workflow job templates, see Work with notifications.

8.4. VIEW COMPLETED WORKFLOW JOBS

The **Jobs** tab provides the list of job templates that have run. Click the expand > icon next to each job to view the details of each job.

From this view, you can click the job ID, name of the workflow job and see its graphical representation. The following example shows the job details of a workflow job:



The nodes are marked with labels to help you identify them. For more information, see the legend in the Workflow visualizer section.

8.5. SCHEDULING A WORKFLOW JOB TEMPLATE

Select the **Schedules** tab to access the schedules for a particular workflow job template..

For more information about scheduling a workflow job template run, see the Scheduling job templates section.

If a workflow job template used in a nested workflow has a survey, or the **Prompt on launch** is selected for the inventory option, the **PROMPT** option displays next to the **SAVE** and **CANCEL** options on the schedule form. Click **PROMPT** to show an optional **INVENTORY** step where you can give or remove an inventory or skip this step without any changes.

8.6. SURVEYS IN WORKFLOW JOB TEMPLATES

Workflows containing job types of **Run** or **Check** provide a way to set up surveys in the workflow job template creation or editing screens.

For more information on job surveys, including how to create a survey and optional survey questions in workflow job templates, see the Surveys in job templates section.

8.7. WORKFLOW VISUALIZER

The Workflow Visualizer provides a graphical way of linking together job templates, workflow templates, project syncs, and inventory syncs to build a workflow template. Before you build a workflow template, see the Workflows section for considerations associated with various scenarios on parent, child, and sibling nodes.

8.7.1. Building a workflow

You can set up any combination of two or more of the following node types to build a workflow:

- Template (Job Template or Workflow Job Template)
- Project Sync
- Inventory Sync
- Approval

Each node is represented by a rectangle while the relationships and their associated edge types are represented by a line (or link) that connects them.

Procedure

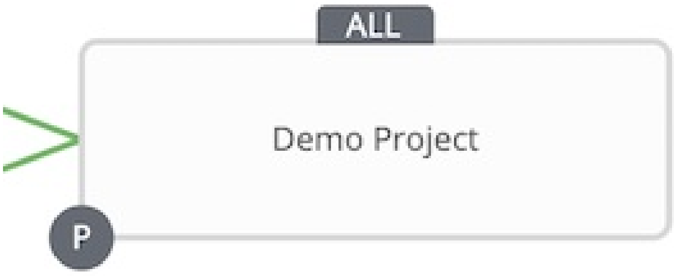
1. To launch the workflow visualizer, use one of these methods:
    - From the navigation panel, select **Automation Execution → Templates**.
        1. Select a workflow template, in the **Details** tab click **Edit template**.
        2. Select the **Visualizer** tab.
    - From the **Templates** list view, click the icon.
2. Click **Start** to display a list of nodes to add to your workflow.
3. From the **Node Type** list, select the type of node that you want to add.
    - If you select an **Approval** node, see Approval nodes for more information.
      Selecting a node provides the available valid options associated with it.

    > **NOTE**
    >
    > If you select a job template that does not have a default inventory when populating a workflow graph, the inventory of the parent workflow is used. Though a credential is not required in a job template, you cannot select a job template for your workflow if it has a credential that requires a password, unless the credential is replaced by a prompted credential.

4. When you select a node type, the workflow begins to build, and you must specify the type of action to be taken for the selected node. This action is also referred to as edge type.
5. If the node is a root node, the edge type defaults to **Always** and is non-editable. For subsequent nodes, you can select one of the following scenarios (edge type) to apply to each:
    - **Always**: Continue to execute regardless of success or failure.
    - **On Success**: After successful completion, execute the next template.
    - **On Failure**: After failure, execute a different template.
6. Select the behavior of the node if it is a convergent node from the **Convergence** field:
    - **Any** is the default behavior, allowing any of the nodes to complete as specified, before triggering the next converging node. If the status of one parent meets one of those run conditions, an **any** child node will run. An **any** node requires all nodes to complete, but only

one node must complete with the expected outcome.

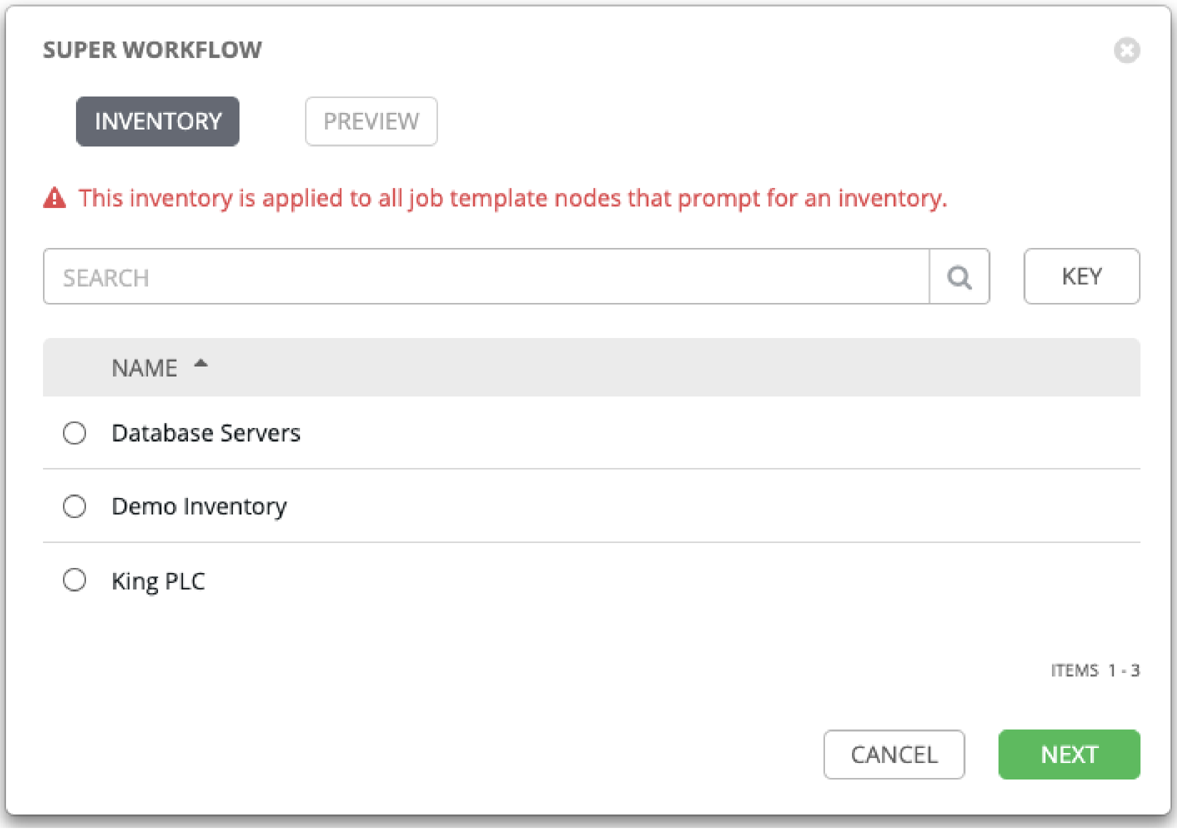
- Choose **All** to ensure that all nodes complete as specified, before converging and triggering the next node. The purpose of **all*** nodes is to make sure that every parent meets its expected outcome to run the child node. The workflow checks to make sure every parent behaves as expected to run the child node. Otherwise, it will not run the child node.
  If selected, the node is labeled as **ALL** in the graphical view:



NOTE

If a node is a root node, or a node that does not have any nodes converging into it, setting the Convergence rule does not apply, as its behavior is dictated by the action that triggers it.

7. If a job template used in the workflow has **Prompt on launch** selected for any of its parameters, a **PROMPT** option appears, enabling you to change those values at the node level. Use the wizard to change the values in each of the tabs and click **Confirm** in the **Preview** tab.
   If a workflow template used in the workflow has **Prompt on launch** selected for the inventory option, use the wizard to supply the inventory at the prompt. If the parent workflow has its own inventory, it overrides any inventory that is supplied here.



SUPER WORKFLOW

⚠ This inventory is applied to all job template nodes that prompt for an inventory.

	NAME ▲
○	Database Servers
○	Demo Inventory
○	King PLC

ITEMS 1 - 3

NOTE

For workflow job templates with required fields that prompt details, but do not have a default, you must give those values when creating a node before the **SELECT** option is enabled.

The following two cases disable the **SELECT** option until a value is provided by the **PROMPT** option:

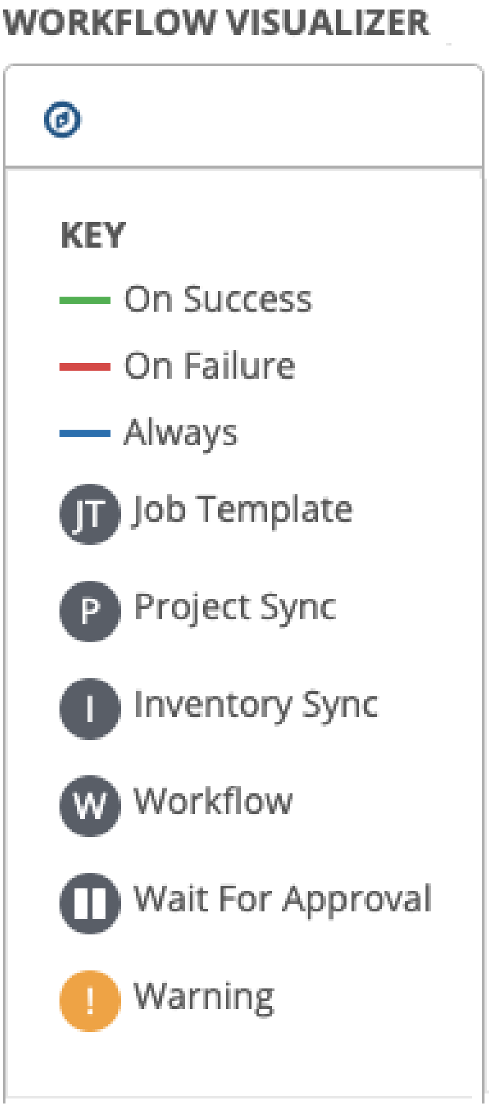
a. When you select the **Prompt on launch** checkbox in a workflow job template, but do not give a default.

b. When you create a survey question that is required but do not give a default answer.

However, this is not the case with credentials. Credentials that require a password on launch are not permitted when creating a workflow node, because everything required to launch the node must be provided when the node is created. If you are prompted for credentials in a workflow job template, it is not possible to select a credential that requires a password in automation controller.

You must also click **SELECT** when the prompt wizard closes, to apply the changes at that node. Otherwise, any changes you make revert back to the values set in the job template.

When the node is created, it is labeled with its job type. A template that is associated with each workflow node runs based on the selected run scenario as it proceeds. Click the compass (🧭) icon to display the legend for each run scenario and their job types.

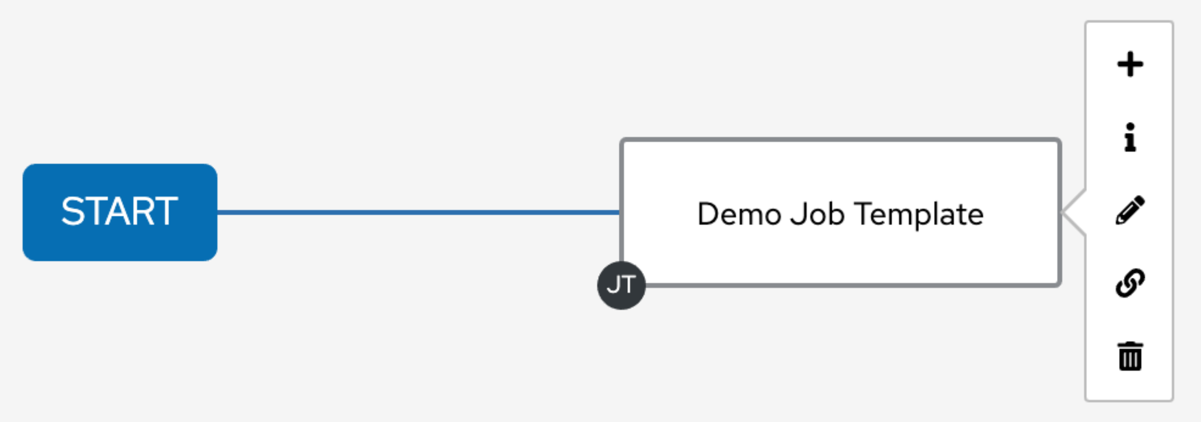


WORKFLOW VISUALIZER

KEY

- On Success
- On Failure
- Always
- JT Job Template
- P Project Sync
- I Inventory Sync
- W Workflow
- Wait For Approval
- ! Warning

8. Hover over a node to add another node, view info about the node, edit the node details, edit an existing link, or delete the selected node:



9. When you have added or edited a node, click **SELECT** to save any modifications and render it on the graphical view. For possible ways to build your workflow, see Building nodes scenarios.

10. When you have built your workflow job template, click **Create workflow job template** to save your entire workflow template and return to the new workflow job template details page.

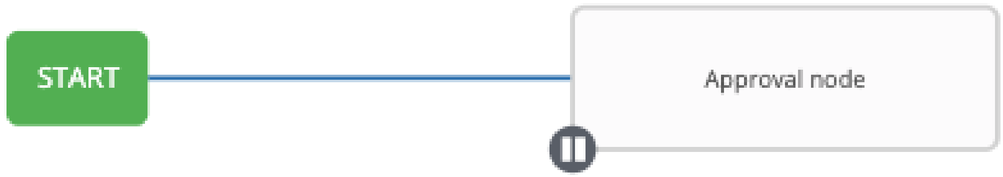
IMPORTANT

Clicking **Close** does not save your work, but instead, it closes the entire Workflow Visualizer so that you have to start again.

8.7.2. Approval nodes

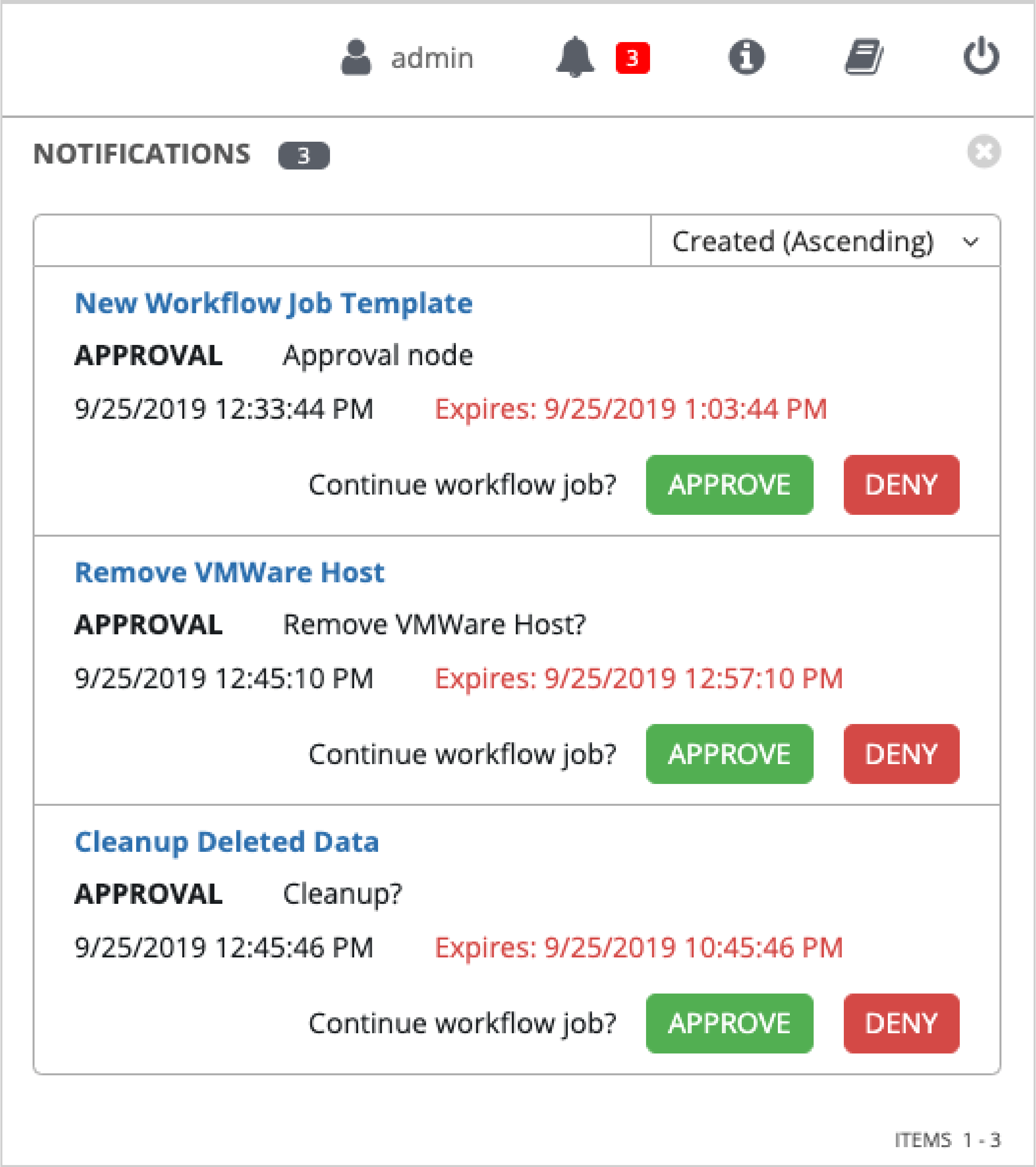
Choosing an **Approval** node requires your intervention in order to advance a workflow. This functions as a means to pause the workflow in between playbooks so that you can give approval to continue on to the next playbook in the workflow. This gives the user a specified amount of time to intervene, but also enables you to continue as quickly as possible without having to wait on another trigger.

The default for the timeout is none, but you can specify the length of time before the request expires and is automatically denied. After you select and supply the information for the approval node, it displays on the graphical view with a pause icon beside it.



The approver is anyone who meets the following criteria:

- A user that can execute the workflow job template containing the approval nodes.
- A user who has organization administrator or above privileges (for the organization associated with that workflow job template).
- A user who has the **Approve** permission explicitly assigned to them within that specific workflow job template.



New Workflow Job Template

APPROVAL Approval node

9/25/2019 12:33:44 PM Expires: 9/25/2019 1:03:44 PM

Continue workflow job?

Remove VMWare Host

APPROVAL Remove VMWare Host?

9/25/2019 12:45:10 PM Expires: 9/25/2019 12:57:10 PM

Continue workflow job?

Cleanup Deleted Data

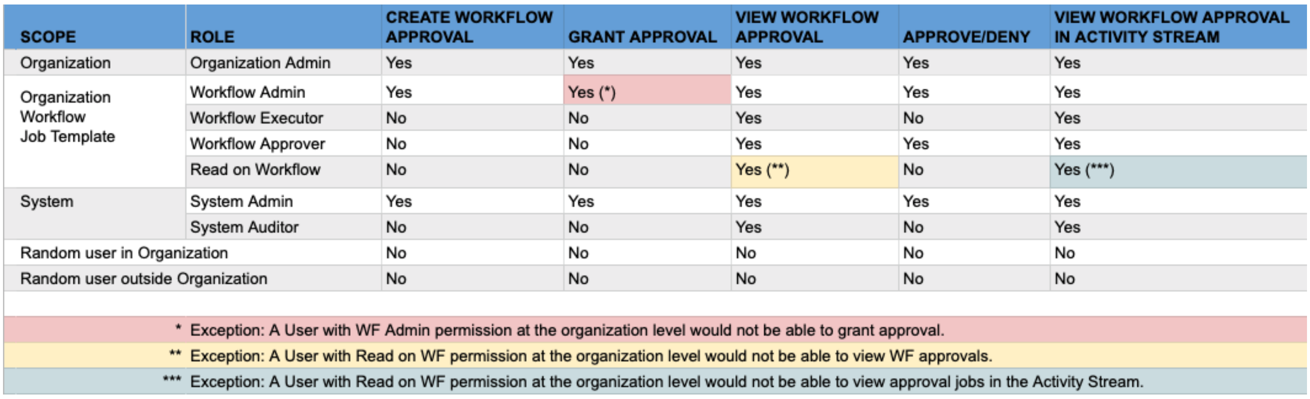
APPROVAL Cleanup?

9/25/2019 12:45:46 PM Expires: 9/25/2019 10:45:46 PM

Continue workflow job?

If pending approval nodes are not approved within the specified time limit (if an expiration was assigned) or they are denied, then they are marked as "timed out" or "failed", and move on to the next "on fail node" or "always node". If approved, the "on success" path is taken. If you try to **POST** in the API to a node that has already been approved, denied or timed out, an error message notifies you that this action is redundant, and no further steps are taken.

The following table shows the various levels of permissions allowed on approval workflows:



SCOPE	ROLE	CREATE WORKFLOW APPROVAL	GRANT APPROVAL	VIEW WORKFLOW APPROVAL	APPROVE/DENY	VIEW WORKFLOW APPROVAL IN ACTIVITY STREAM
Organization	Organization Admin	Yes	Yes	Yes	Yes	Yes
Organization Workflow Job Template	Workflow Admin	Yes	Yes (*)	Yes	Yes	Yes
	Workflow Executor	No	No	Yes	No	Yes
	Workflow Approver	No	No	Yes	Yes	Yes
	Read on Workflow	No	No	Yes (**)	No	Yes (***)
System	System Admin	Yes	Yes	Yes	Yes	Yes
	System Auditor	No	No	Yes	No	Yes
Random user in Organization		No	No	No	No	No
Random user outside Organization		No	No	No	No	No

* Exception: A User with WF Admin permission at the organization level would not be able to grant approval.

** Exception: A User with Read on WF permission at the organization level would not be able to view WF approvals.

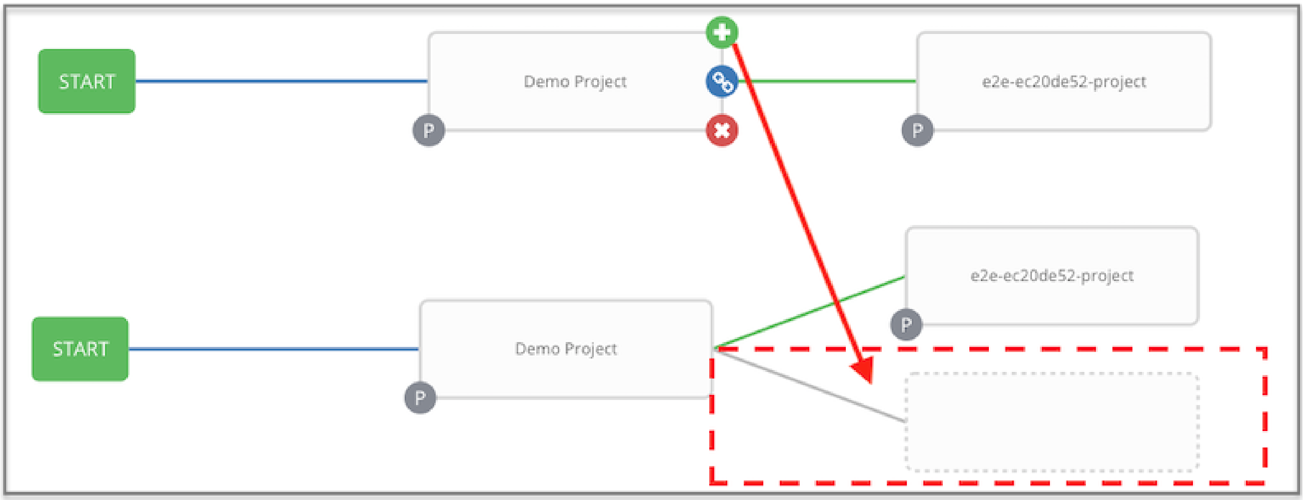
*** Exception: A User with Read on WF permission at the organization level would not be able to view approval jobs in the Activity Stream.

8.7.3. Building nodes scenarios

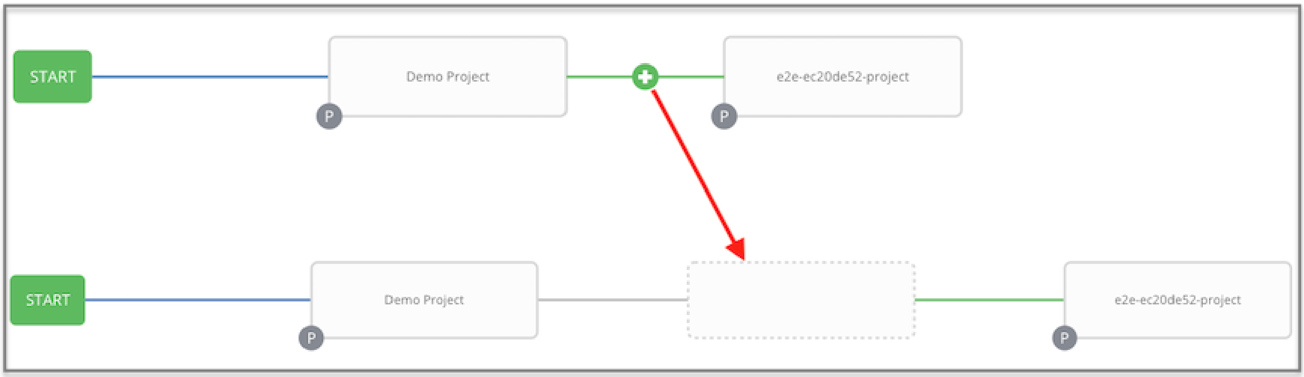
Learn how to manage nodes in the following scenarios.

Procedure

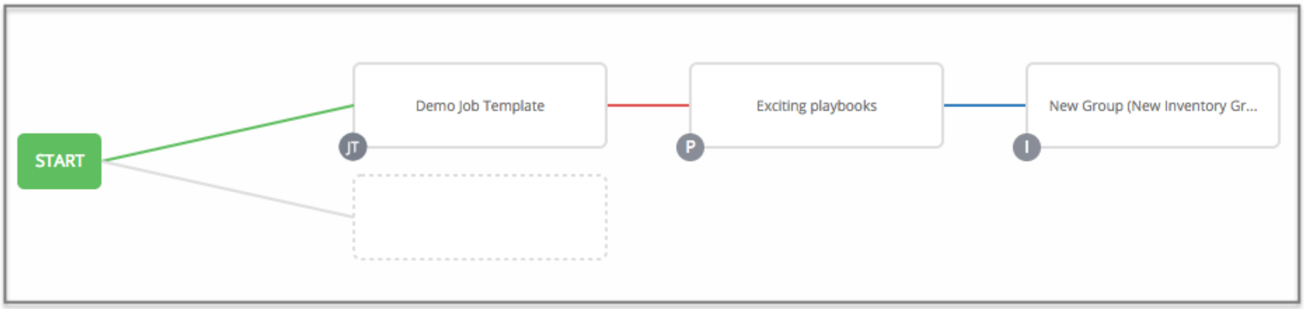
- Click the (+) icon on the parent node to add a sibling node:



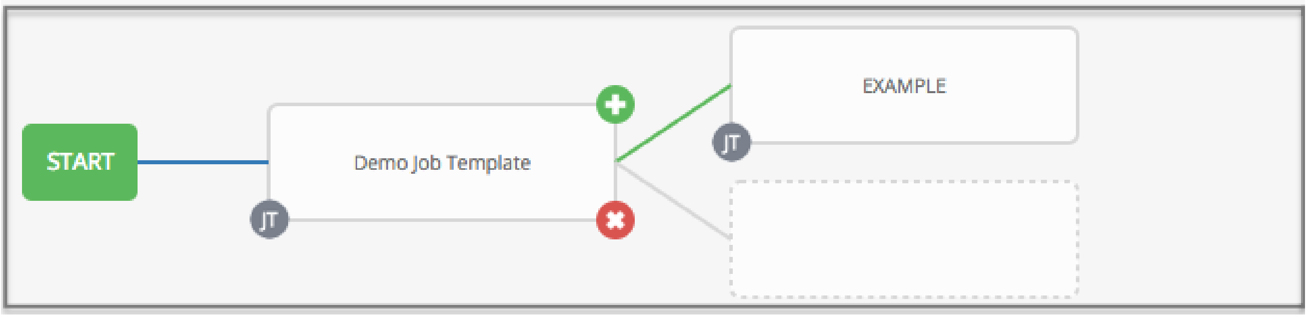
- Hover over the line that connects two nodes and click the plus (+), to insert another node in between nodes. Clicking the plus (+) icon automatically inserts the node between the two nodes:



- Click **START** again, to add a root node to depict a split scenario:



- At any node where you want to create a split scenario, hover over the node from which the split scenario begins and click the plus (+) icon. This adds multiple nodes from the same parent node, creating sibling nodes:

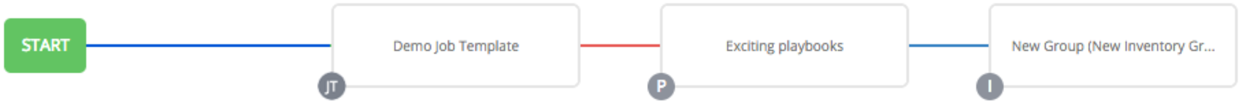


NOTE

When adding a new node, the **PROMPT** option also applies to workflow templates. Workflow templates prompt for inventory and surveys.

- You can undo the last inserted node by using one of these methods:
  - Click on another node without making a selection.
  - Click **Cancel**.

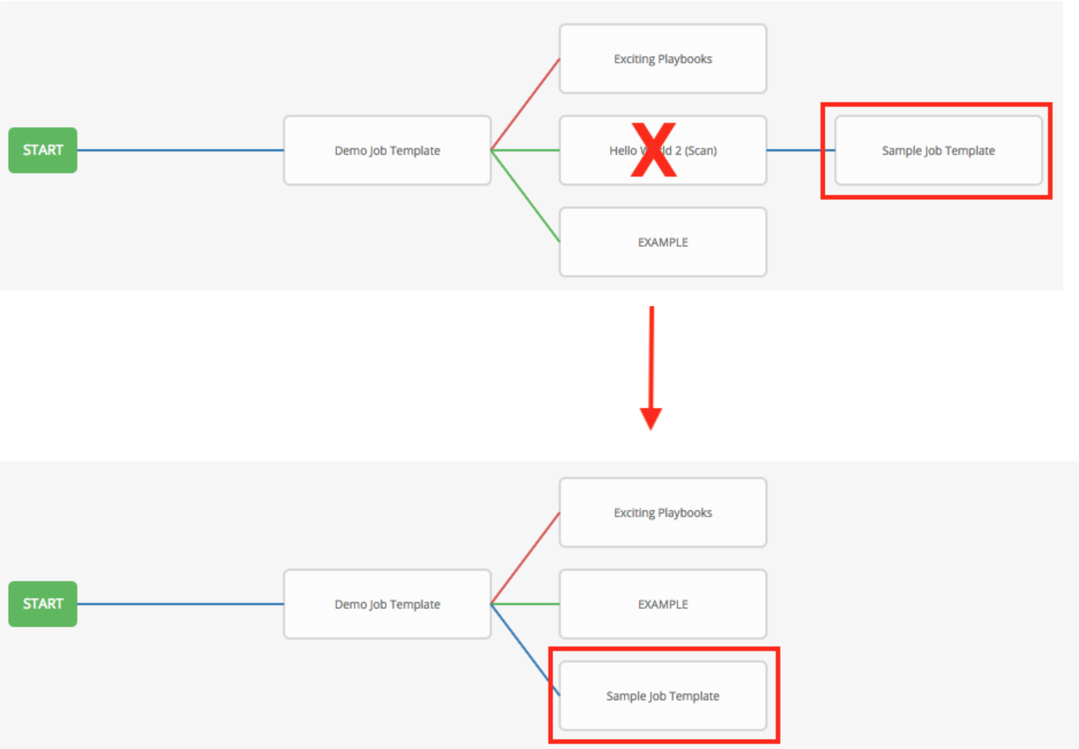
The following example workflow contains all three types of jobs initiated by a job template. If it fails to run, you must protect the sync job. Regardless of whether it fails or succeeds, proceed to the inventory sync job:



Refer to the key by clicking the compass (🧭) icon to identify the meaning of the symbols and colors associated with the graphical depiction.

NOTE

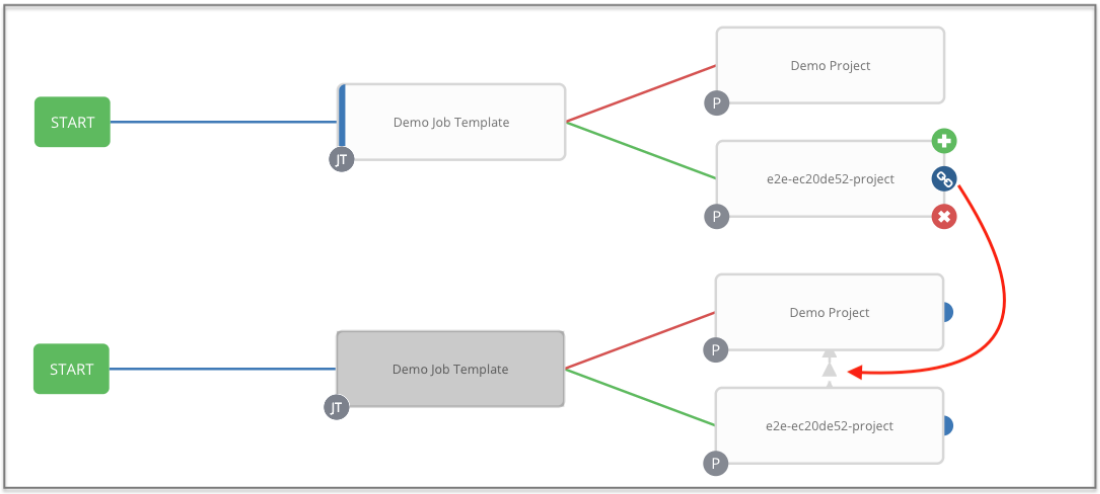
If you remove a node that has a follow-on node attached to it in a workflow with a set of sibling nodes that has varying edge types, the attached node automatically joins the set of sibling nodes and retains its edge type:



8.7.4. Editing a node

Procedure

- Edit a node by using one of these methods:
  - If you want to edit a node, click on the node you want to edit. The pane displays the current selections. Make your changes and click **Select** to apply them to the graphical view.
  - To edit the edge type for an existing link, (**success**, **failure**, **always**), click the link. The pane displays the current selection. Make your changes and click **Save** to apply them to the graphical view.
  - Click the link (🔗) icon that appears on each node, to add a new link from one node to another. Doing this highlights the nodes that are possible to link to. These options are indicated by the dotted lines. Invalid options are indicated by disabled boxes (nodes) that would otherwise produce an invalid link. The following example shows the **Demo Project** as a possible option for the **e2e-ec20de52-project** to link to, indicated by the arrows:



   - To remove a link, click the link and click **UNLINK**. This option only appears in the pane if the target or child node has more than one parent. All nodes must be linked to at least one other node at all times so you must create a new link before removing an old one.

- Edit the view of the workflow diagram by using one of these methods:
   - Click the settings icon to zoom, pan, or reposition the view.
   - Drag the workflow diagram to reposition it on the screen or use the scroll on your mouse to zoom.

8.8. LAUNCHING A WORKFLOW JOB TEMPLATE

Procedure

- Launch a workflow job template by using one of these methods:
   - From the navigation panel, select **Automation Execution → Templates** and click the 🚀 icon next to the job template.
   - Click **Launch template** in the **Details** tab of the workflow job template that you want to launch.

Variables added for a workflow job template are automatically added in automation controller when launching, along with any extra variables set in the workflow job template and survey.

Events related to approvals on workflows are displayed in the activity stream (⟲) with detailed information about the approval requests, if any.

8.9. COPYING A WORKFLOW JOB TEMPLATE

With automation controller you can copy a workflow job template. When you copy a workflow job template, it does not copy any associated schedule, notifications, or permissions. Schedules and notifications must be recreated by the user or system administrator creating the copy of the workflow template. The user copying the workflow template is granted the administrator permission, but no permissions are assigned (copied) to the workflow template.

Procedure

1. Open the workflow job template that you want to copy by using one of these methods:
   - From the navigation panel, select **Automation Execution** → **Templates**.
   - In the workflow job template **Details** view, click ⋮ next to the desired template.
     - Click the copy (⧉) icon.
       The new template with the name of the template from which you copied and a timestamp displays in the list of templates.
2. Select the copied template and click **Edit template**.
3. Replace the contents of the **Name** field with a new name, and give or change the entries in the other fields to complete this template.
4. Click **Save job template**.

NOTE

If a resource has a related resource that you do not have the right level of permission to, you cannot copy the resource. For example, in the case where a project uses a credential that a current user only has Read access. However, for a workflow job template, if any of its nodes use an unauthorized job template, inventory, or credential, the workflow template can still be copied. But in the copied workflow job template, the corresponding fields in the workflow template node are absent.

8.10. WORKFLOW JOB TEMPLATE EXTRA VARIABLES

For more information see the Extra variables section.

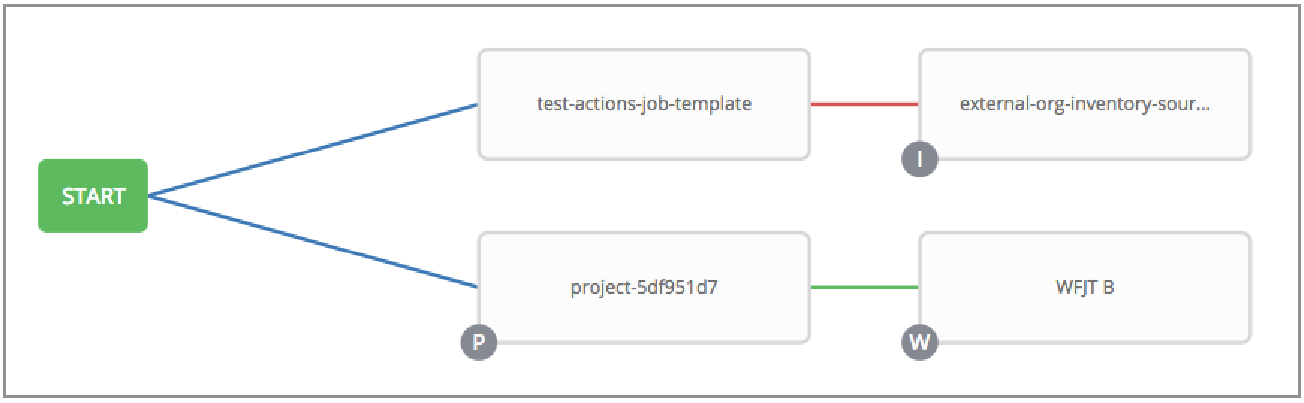
CHAPTER 9. WORKFLOWS IN AUTOMATION CONTROLLER

Workflows enable you to configure a sequence of disparate job templates (or workflow templates) that may or may not share inventory, playbooks, or permissions.

Workflows have **admin** and **execute** permissions, similar to job templates. A workflow accomplishes the task of tracking the full set of jobs that were part of the release process as a single unit.

Job or workflow templates are linked together using a graph-like structure called nodes. These nodes can be jobs, project syncs, or inventory syncs. A template can be part of different workflows or used multiple times in the same workflow. A copy of the graph structure is saved to a workflow job when you launch the workflow.

The following example shows a workflow that contains all three, as well as a workflow job template:

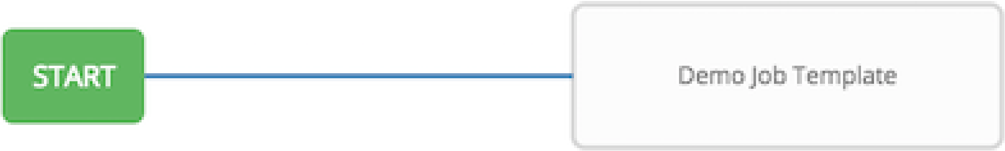


As the workflow runs, jobs are spawned from the node’s linked template. Nodes linking to a job template which has prompt-driven fields (job_type, job_tags, skip_tags, limit) can contain those fields, and is not prompted on launch. Job templates that prompt for a credential or inventory, without defaults, are not available for inclusion in a workflow.

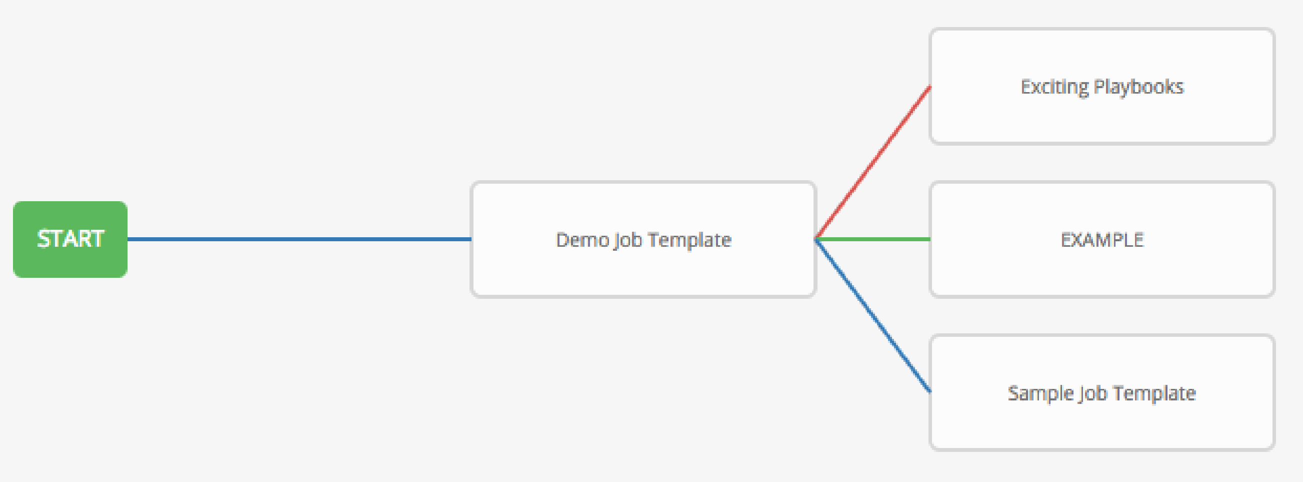
9.1. WORKFLOW SCENARIOS AND CONSIDERATIONS

When building workflows, consider the following:

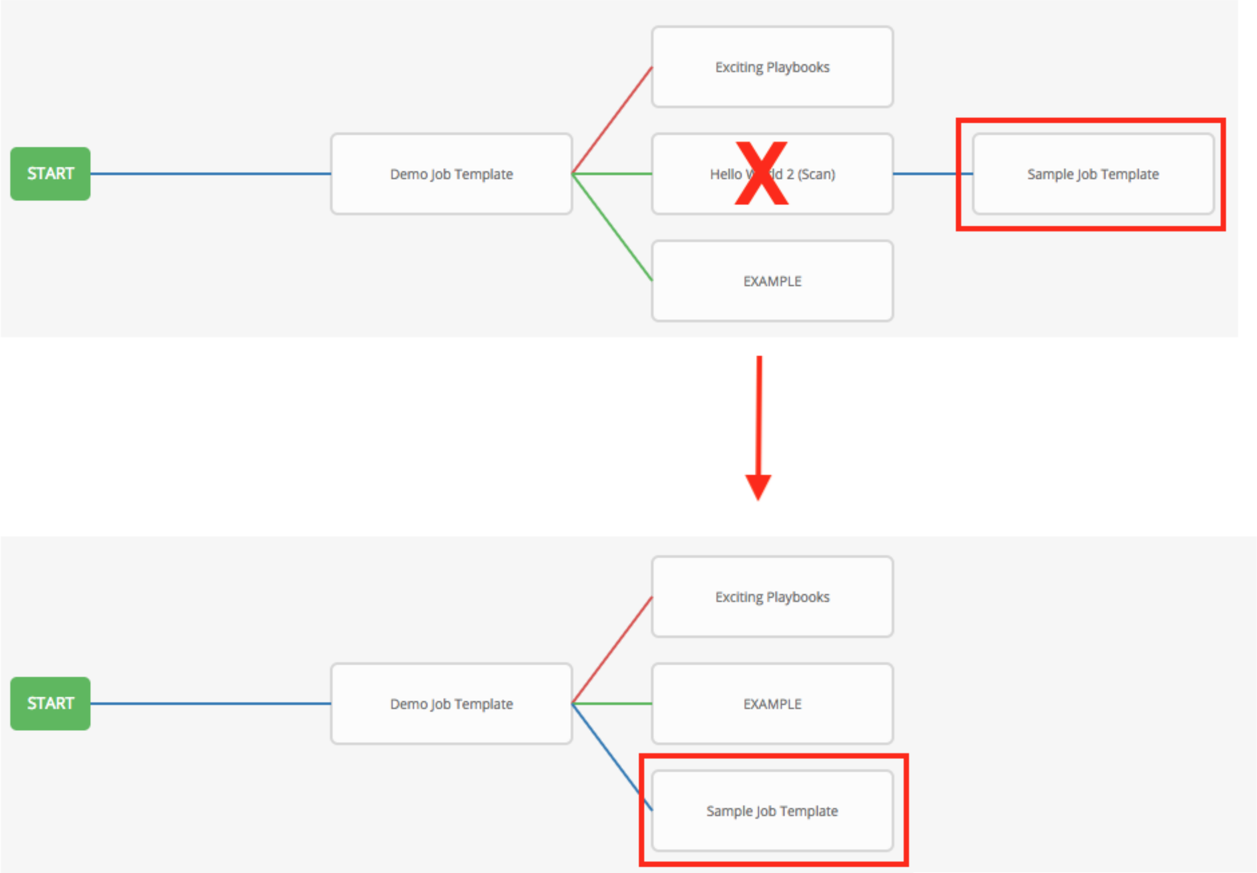
- A root node is set to **ALWAYS** by default and cannot be edited.



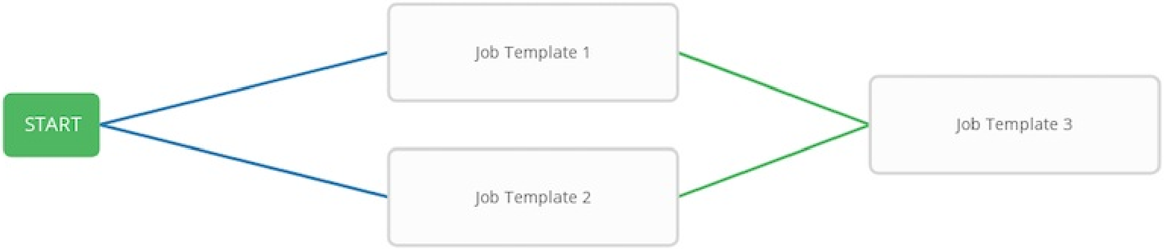
- A node can have multiple parents, and children can be linked to any of the states of success, failure, or always. If always, then the state is neither success nor failure. States apply at the node level, not at the workflow job template level. A workflow job is marked as successful unless it is canceled or encounters an error.



- If you remove a job or workflow template within the workflow, the nodes previously connected to those deleted, automatically get connected upstream and retain the edge type as in the following example:

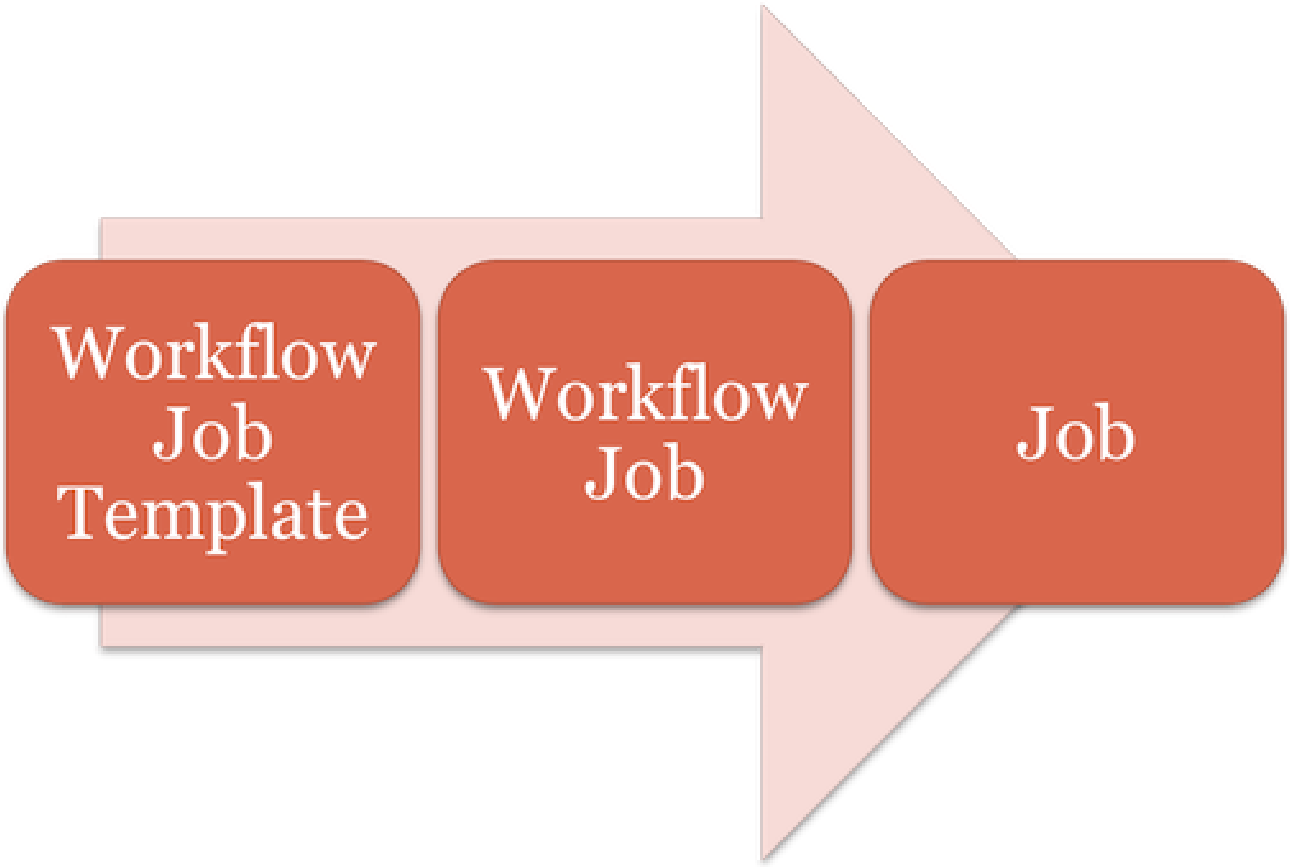


- You can have a convergent workflow, where multiple jobs converge into one. In this scenario, any of the jobs or all of them must complete before the next one runs, as shown in the following example:



- In this example, automation controller runs the first two job templates in parallel. When they both finish and succeed as specified, the third downstream (convergence node), triggers.

- Prompts for inventory and surveys apply to workflow nodes in workflow job templates.
- If you launch from the API, running a **get** command displays a list of warnings and highlights missing components. The following image illustrates a basic workflow for a workflow job template:



- It is possible to launch several workflows simultaneously, and set a schedule for when to launch them. You can set notifications on workflows, such as when a job completes, similar to that of job templates.

NOTE

Job slicing is intended to scale job executions horizontally.

If you enable job slicing on a job template, it divides the inventory to be acted on in the number of slices configured at launch time. Then starts a job for each slice.

For more information see the Job slicing section.

- You can build a recursive workflow, but if automation controller detects an error, it stops at the time the nested workflow attempts to run.
- Artifacts gathered in jobs in the sub-workflow are passed to downstream nodes.
- An inventory can be set at the workflow level, or prompt for inventory on launch.
- When launched, all job templates in the workflow that have **ask_inventory_on_launch=true** use the workflow level inventory.
- Job templates that do not prompt for inventory ignore the workflow inventory and run against their own inventory.
- If a workflow prompts for inventory, schedules and other workflow nodes can provide the inventory.
- In a workflow convergence scenario, **set_stats** data is merged in an undefined way, therefore you must set unique keys.

9.2. WORKFLOW EXTRA VARIABLES

Workflows use surveys to specify variables to be used in the playbooks in the workflow, called **extra_vars**. Survey variables are combined with **extra_vars** defined on the workflow job template, and saved to the workflow job **extra_vars**. **extra_vars** in the workflow job are combined with job template variables when spawning jobs within the workflow.

Workflows use the same behavior (hierarchy) of variable precedence as job templates with the exception of three additional variables. See the Automation controller Variable Precedence Hierarchy in the Extra variables section of Job templates. The three additional variables include:

- Workflow job template extra variables
- Workflow job template survey (defaults)
- Workflow job launch extra variables

Workflows included in a workflow follow the same variable precedence, they only inherit variables if they are specifically prompted for, or defined as part of a survey.

In addition to the workflow **extra_vars**, jobs and workflows run as part of a workflow can inherit variables in the artifacts dictionary of a parent job in the workflow (also combining with ancestors further upstream in its branch). These can be defined by the **set_stats** Ansible module.

If you use the **set_stats** module in your playbook, you can produce results that can be consumed downstream by another job.

Example

Notifying users as to the success or failure of an integration run. In this example, there are two playbooks that can be combined in a workflow to exercise artifact passing:

- invoke_set_stats.yml: first playbook in the workflow:

```
---
- hosts: localhost
  tasks:
    - name: "Artifact integration test results to the web"
      local_action: 'shell curl -F "file=@integration_results.txt" https://file.io'
      register: result


    - name: "Artifact URL of test results to Workflows"
      set_stats:
        data:
          integration_results_url:  "{{ (result.stdout|from_json).link }}"
```

- use_set_stats.yml: second playbook in the workflow:

```
---
- hosts: localhost
  tasks:
    - name: "Get test results from the web"
      uri:
        url: "{{ integration_results_url }}"
        return_content: true
      register: results


    - name: "Output test results"
      debug:
        msg: "{{ results.content }}"
```

The **set_stats** module processes this workflow as follows:

1. The contents of an integration result is uploaded to the web.
2. Through the **invoke_set_stats** playbook, **set_stats** is then invoked to artifact the URL of the uploaded **integration_results.txt** into the Ansible variable "integration_results_url".
3. The second playbook in the workflow consumes the Ansible extra variable "integration_results_url". It calls out to the web using the uri module to get the contents of the file uploaded by the previous job template job. Then, it prints out the contents of the obtained file.

NOTE

For artifacts to work, keep the default setting, **per_host = False** in the **set_stats** module.

9.3. WORKFLOW STATES

The workflow job can have the following states (no Failed state):

- Waiting

- Running
- Success (finished)
- Cancel
- Error
- Failed

In the workflow scheme, canceling a job cancels the branch, while canceling the workflow job cancels the entire workflow.

9.4. ROLE-BASED ACCESS CONTROLS

To edit and delete a workflow job template, you must have the administrator role. To create a workflow job template, you must be an organization administrator or a system administrator. However, you can run a workflow job template that contains job templates that you do not have permissions for. System administrators can create a blank workflow and then grant an **admin_role** to a low-level user, after which they can delegate more access and build the graph. You must have **execute** access to a job template to add it to a workflow job template.

You can also perform other tasks, such as making a duplicate copy or re-launching a workflow, depending on which permissions are granted to a user. You must have permissions to all the resources used in a workflow, such as job templates, before relaunching or making a copy.

For more information, see Managing access with role based access control .

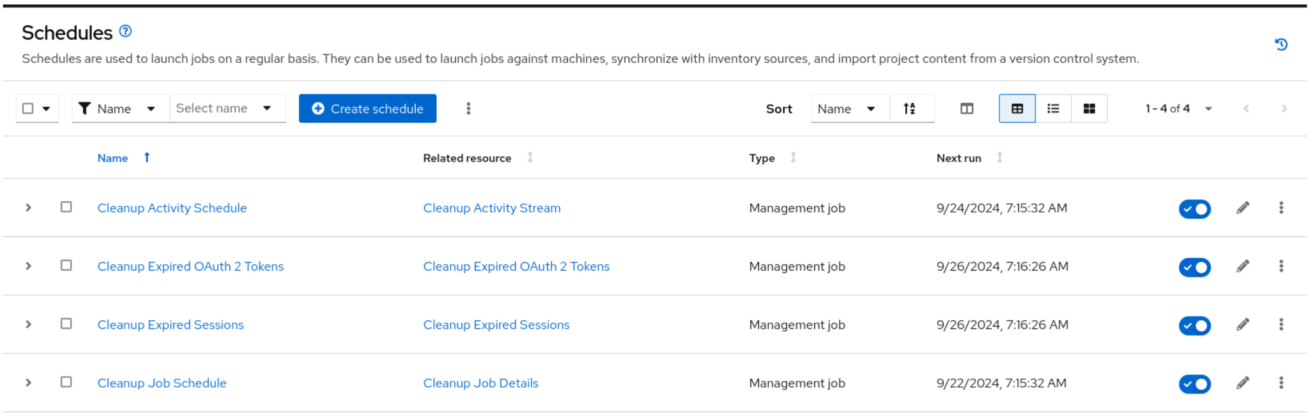
For more information about performing the tasks described, see Workflow job templates.

CHAPTER 10. SCHEDULES

From the navigation panel, click **Automation Execution → Schedules** to access your configured schedules. The schedules list can be sorted by any of the attributes from each column using the directional arrows. You can also search by name, date, or the name of the month in which a schedule runs.

Use the **On** or **Off** toggle to stop an active schedule or activate a stopped schedule.

Click the Edit ✏ icon to edit a schedule.



Schedules

Schedules are used to launch jobs on a regular basis. They can be used to launch jobs against machines, synchronize with inventory sources, and import project content from a version control system.

Name	Related resource	Type	Next run
Cleanup Activity Schedule	Cleanup Activity Stream	Management job	9/24/2024, 7:15:32 AM
Cleanup Expired OAuth 2 Tokens	Cleanup Expired OAuth 2 Tokens	Management job	9/26/2024, 7:16:26 AM
Cleanup Expired Sessions	Cleanup Expired Sessions	Management job	9/26/2024, 7:16:26 AM
Cleanup Job Schedule	Cleanup Job Details	Management job	9/22/2024, 7:15:32 AM

If you are setting up a template, a project, or an inventory source, click the **Schedules** tab on the **Details** page for that resource, to configure schedules for these resources. When you create a schedule, it has the following parameters:

Name

Click the schedule name to open its details.

Related resource

Describes the function of the schedule.

Type

This identifies whether the schedule is associated with a source control update or a system-managed job schedule.

Next run

The next scheduled run of this task.

10.1. ADDING A NEW SCHEDULE

You can create schedules from a template, project, or inventory source, and directly on the main **Schedules** page.

To create a new schedule on the **Schedules** page:

Procedure

1. From the navigation panel, select **Automation Execution → Schedules**.
2. Click **Create schedule**. This opens the **Create Schedule** window.
3. Select a **Resource type** onto which this schedule is applied.
   Select from:

- **Job template**
  - For **Job template** select a **Job template** from the menu.
- **Workflow job template**
  - For **Workflow job template** select a **Workflow job template** from the menu.
- **Inventory source**
  - For **Inventory source** select an **Inventory** and an **Inventory source** from the appropriate menu.
- **Project sync**
  - For **Project sync** select a **Project** from the menu.
- **Management job template**
  - For **Management job template** select a **Workflow job template** from the menu.

To create a new schedule from a resource page:

Procedure

1. Click the **Schedules** tab of the resource that you are configuring. This can be a template, project, or inventory source.
2. Click **Create schedule**. This opens the **Create Schedule** window.

For both procedures

1. For **Job template** and **Project sync** enter the appropriate details into the following fields:
   - **Schedule name**: Enter the name.
   - Optional: **Description**: Enter a description.
   - **Start date/time**: Enter the date and time to start the schedule.
   - **Time zone**: Select the time zone. The **Start date/time** that you enter must be in this time zone.
     The **Schedule Details** display when you establish a schedule, enabling you to review the schedule settings and a list of the scheduled occurrences in the selected **Local Time Zone**.

     IMPORTANT

     Jobs are scheduled in UTC. Repeating jobs that run at a specific time of day can move relative to a local time zone when Daylight Savings Time shifts occur. The system resolves the local time zone based time to UTC when the schedule is saved. To ensure your schedules are correctly created, set your schedules in UTC time.
2. Click **Next**. The **Define rules** page is displayed.

10.1.1. Defining rules for the schedule

Enter the following information:

- **Frequency**: Enter how frequently the schedule runs.
- **Interval**:
- **Week Start**: Select the day of the week that you want the week to begin.
- **Weekdays**: Select the days of the week on which to run the schedule.
- **Months**: Select the months of the year on which to run the schedule
- **Annual week(s) number**: This field is used to declare numbered weeks of the year that the schedule should run.
- **Minute(s) of hour**: This field is used to declare minute(s) of the hour that the schedule should run.
- **Hour of day**: This field is used to declare the hours of day that the schedule should run.
- **Monthly day(s) number**: This field is used to declare ordinal days number of the month that the schedule should run.
- **Annual day(s) number**: This field is used to declare ordinal number days of the year that the schedule should run.
- **Occurences**: Use this field to filter down indexed rules based on those declared using the form fields in the Rule section.
  For more information, see the link to the **iCalendar RFC for bysetpos** field in the iCalendar documentation when you have set the rules for the schedule.
- **Count**: The number of times this rule should be used.
- **Until**: Use this rule until the specified date and time

Click **Save rule** The **Schedule Rules** summary page is displayed.

Click **Add rule** to add additional rules. Click **Next**.

The **Schedule Exceptions** summary page is displayed.

10.1.2. Setting exceptions to the schedule

On the **Create Schedule** page, click **Create exception**.

Use the same format as for the schedule rules to create a schedule exception.

Click **Next** to save and review both the schedule and the exception.

CHAPTER 11. PROJECTS

A Project is a logical collection of Ansible playbooks, represented in automation controller. You can manage playbooks and playbook directories different ways:

- By placing them manually under the Project Base Path on your automation controller server.
- By placing your playbooks into a source code management (SCM) system supported by the automation controller. These include Git, Subversion, Mercurial and Red Hat Insights.

For more information on creating a Red Hat Insights project, see Setting up Red Hat Insights for Red Hat Ansible Automation Platform Remediations.

> NOTE
>
> The Project Base Path is **/var/lib/awx/projects**. However, this can be modified by the system administrator. It is configured in **/etc/tower/conf.d/custom.py**.
>
> Use caution when editing this file, as incorrect settings can disable your installation.

The Projects page displays the list of the projects that are currently available.

A **Demo Project** is provided that you can work with initially.

The default view is collapsed (**Compact**) with project name and its status, but you can use the ❯ next to each entry to expand for more information.

For each project listed, you can get the latest SCM revision ⟳, edit ✎ the project, or copy ⧉ the project attributes, using the icons next to each project.

Projects can be updated while a related job is running.

In cases where you have a large project (around 10 GB), disk space on **/tmp** may be an issue.

Status indicates the state of the project and may be one of the following (note that you can also filter your view by specific status types):

- **Pending** - The source control update has been created, but not queued or started yet. Any job (not just source control updates) stays in pending until it is ready to be run by the system. Possible reasons for it not being ready are:
  - It has dependencies that are currently running so it has to wait until they are done.
  - There is not enough capacity to run in the locations it is configured to.
- **Waiting** - The source control update is in the queue waiting to be executed.
- **Running** - The source control update is currently in progress.
- **Successful** - The last source control update for this project succeeded.
- **Failed** - The last source control update for this project failed.
- **Error** - The last source control update job failed to run at all.
- **Canceled** - The last source control update for the project was canceled.

- **Never updated** - The project is configured for source control, but has never been updated.
- **OK** - The project is not configured for source control, and is correctly in place.
- **Missing** - Projects are absent from the project base path of **/var/lib/awx/projects**. This is applicable for manual or source control managed projects.

NOTE

Projects of credential type **Manual** cannot update or schedule source control-based actions without being reconfigured as an SCM type credential.

11.1. ADDING A NEW PROJECT

You can create a logical collection of playbooks, called projects in automation controller.

Procedure

1. From the navigation panel, select **Automation Execution → Projects**.
2. On the **Projects** page, click **Create project** to launch the **Create Project** window.
3. Enter the appropriate details into the following required fields:
    - **Name** (required)
    - Optional: **Description**
    - **Organization** (required): A project must have at least one organization. Select one organization now to create the project. When the project is created you can add additional organizations.
    - Optional: **Execution environment**: Enter the name of the execution environment or search from a list of existing ones to run this project. For more information, see Creating and using execution environments.
    - **Source control type** (required): Select an SCM type associated with this project from the menu. Options in the following sections become available depending on the type chosen. For more information, see Managing playbooks manually or Managing playbooks using source control.
    - Optional: **Content signature validation credential**: Use this field to enable content verification. Specify the GPG key to use for validating content signature during project synchronization. If the content has been tampered with, the job will not run. For more information, see Project signing and verification .
4. Click **Create project**.

Additional resources

The following describe the ways projects are sourced:

- Managing playbooks manually
- Managing playbooks using source control
    - SCM Types - Configuring playbooks to use Git and Subversion

  - SCM Types - Configuring playbooks to use Git and Subversion
  - SCM Type - Configuring playbooks to use Red Hat Insights
  - SCM Type - Configuring playbooks to use a remote archive

11.1.1. Managing playbooks manually

Procedure

- Create one or more directories to store playbooks under the Project Base Path, for example, **/var/lib/awx/projects/**.
- Create or copy playbook files into the playbook directory.
- Ensure that the playbook directory and files are owned by the same UNIX user and group that the service runs as.
- Ensure that the permissions are appropriate for the playbook directories and files.

Troubleshooting

- If you have not added any Ansible Playbook directories to the base project path an error message is displayed. Choose one of the following options to troubleshoot this error:
  - Create the appropriate playbook directories and check out playbooks from your (Source code management) SCM.
  - Copy playbooks into the appropriate playbook directories.

11.1.2. Managing playbooks using source control

Choose one of the following options when managing playbooks using source control:

- SCM Types - Configuring playbooks to use Git and Subversion
- SCM Type - Configuring playbooks to use Red Hat Insights
- SCM Type - Configuring playbooks to use a remote archive

11.1.2.1. SCM Types - Configuring playbooks to use Git and Subversion

Procedure

1. From the navigation panel, select **Automation Execution → Projects**.
2. Click the project name you want to use.
3. In the project **Details** tab, click **Edit project**.
4. Select the appropriate option (Git or Subversion) from the **Source control type** menu.
5. Enter the appropriate details into the following fields:
   - **Source control URL** - See an example in the tooltip .

- Optional: **Source control branch/tag/commit**: Enter the SCM branch, tags, commit hashes, arbitrary refs, or revision number (if applicable) from the source control (Git or Subversion) to checkout. Some commit hashes and references might not be available unless you also give a custom refspec in the next field. If left blank, the default is **HEAD** which is the last checked out Branch, Tag, or Commit for this project.

- **Source control refspec** - This field is an option specific to git source control and only advanced users familiar and comfortable with git should specify which references to download from the remote repository. For more information, see Job branch overriding.

- **Source control credential** - If authentication is required, select the appropriate source control credential.

6. Optional: **Options** - select the launch behavior, if applicable:

    - **Clean** - Removes any local modifications before performing an update.

    - **Delete** - Deletes the local repository in its entirety before performing an update. Depending on the size of the repository this can significantly increase the amount of time required to complete an update.

    - **Track submodules** - Tracks the latest commit. There is more information in the tooltip ? .

    - **Update revision on launch** - Updates the revision of the project to the current revision in the remote source control, and caching the roles directory from Galaxy or Collections support. Automation controller ensures that the local revision matches and that the roles and collections are up-to-date with the last update. In addition, to avoid job overflows if jobs are spawned faster than the project can synchronize, selecting this enables you to configure a Cache Timeout to cache previous project synchronizations for a given number of seconds.

    - **Allow branch override** - Enables a job template or an inventory source that uses this project to start with a specified SCM branch or revision other than that of the project. For more information, see Job branch overriding.

7. Click **Save project**.

TIP

Using a GitHub link is an easy way to use a playbook. To help get you started, use the **helloworld.yml** file available here.

11.1.2.2. SCM Type - Configuring playbooks to use Red Hat Insights

Procedure

1. From the navigation panel, select **Automation Execution** → **Projects**.

2. Click the project name you want to use.

3. In the project **Details** tab, click **Edit project**.

4. Select **Red Hat Insights** from the **Source Control Type** menu.

5. In the **Insights credential** field, select the appropriate credential for use with Insights, as Red Hat Insights requires a credential for authentication.

6. Optional: In the **Options** field, select the launch behavior, if applicable:
   - **Clean** - Removes any local modifications before performing an update.
   - **Delete** - Deletes the local repository in its entirety before performing an update. Depending on the size of the repository this can significantly increase the amount of time required to complete an update.
   - **Update revision on launch** - Updates the revision of the project to the current revision in the remote source control, and caches the roles directory from Ansible Galaxy support or Collections support. Automation controller ensures that the local revision matches, and that the roles and collections are up-to-date. If jobs are spawned faster than the project can synchronize, selecting this enables you to configure a Cache Timeout to cache previous project synchronizations for a certain number of seconds, to avoid job overflow.
7. Click **Save project**.

11.1.2.3. SCM Type - Configuring playbooks to use a remote archive

Playbooks that use a remote archive enable projects to be based on a build process that produces a versioned artifact, or release, containing all the requirements for that project in a single archive.

Procedure

1. From the navigation panel, select **Automation Execution → Projects**.
2. Click the project name you want to use.
3. In the project **Details** tab, click **Edit project**.
4. Select **Remote Archive** from the **Source control type** menu.
5. Enter the appropriate details into the following fields:
   - **Source control URL** - requires a URL to a remote archive, such as a *GitHub Release* or a build artifact stored in *Artifactory* and unpacks it into the project path for use.
   - **Source control credential** - If authentication is required, select the appropriate source control credential.
6. Optional: In the **Options** field, select the launch behavior, if applicable:
   - **Clean** - Removes any local modifications before performing an update.
   - **Delete** - Deletes the local repository in its entirety before performing an update. Depending on the size of the repository this can significantly increase the amount of time required to complete an update.
   - **Update revision on launch** - Not recommended. This option updates the revision of the project to the current revision in the remote source control, and caches the roles directory from Ansible Galaxy support or Collections support.
   - **Allow branch override** - Not recommended. This option enables a job template that uses this project to launch with a specified SCM branch or revision other than that of the project’s.

NOTE

Since this source control type is intended to support the concept of unchanging artifacts, it is advisable to disable Galaxy integration (for roles, at a minimum).

7. Click **Save project**.

11.2. UPDATING PROJECTS FROM SOURCE CONTROL

Procedure

1. From the navigation panel, select **Automation Execution → Projects**.
2. Click the sync 🔄 icon next to the project that you want to update.

NOTE

Immediately after adding a project setup to use source control, a sync starts that fetches the project details from the configured source control.

- Click the project’s status under the **Status** column for further information about the update process. This brings you to the **Output** tab of the **Jobs** section.

11.3. WORK WITH PERMISSIONS

The set of permissions assigned to a project (role-based access controls) that provide the ability to read, change, and administer projects, inventories, job templates, and other elements are privileges.

To access the project permissions, select the **Access** tab of the **Projects** page. This screen displays a list of users that currently have permissions to this project.

You can sort and search this list by **Username**, **First Name**, or **Last Name**.

11.3.1. Adding project permissions

Manage the permissions that users and teams have to access a project.

Procedure

1. From the navigation panel, select **Automation Execution → Projects**.
2. Select the project that you want to update and click the **User Access** or **Team Access** tab.
3. Click **Add roles**.
4. Select a user or team to add and click **Next**.
5. Select one or more users or teams from the list by clicking the checkbox next to the name to add them as members.
6. Click **Next**.

7. Select the roles you want the selected users or teams to have. Different resources have different options available.

8. Click **Finish** to apply the roles to the selected users or teams and to add them as members. The updated roles assigned for each user and team are displayed.

11.3.2. Removing permissions from a project

Remove roles for a particular user.

Procedure

1. From the navigation panel, select **Automation Execution → Projects**.

2. Select the project that you want to update and click the **User Access** or **Team Access** tab.

3. Click the ✕ icon next to the user role in the **Roles** column.

4. Click **Delete** in the confirmation window to confirm the disassociation.

11.4. ANSIBLE GALAXY SUPPORT

At the end of a project update, automation controller searches for the **requirements.yml** file in the **roles** directory, located at **<project-top-level-directory>/roles/requirements.yml**.

If this file is found, the following command automatically runs:

```
ansible-galaxy role install -r roles/requirements.yml -p <project-specific cache
location>/requirements_roles -vvv
```

This file enables you to reference Ansible Galaxy roles or roles within other repositories which can be checked out in conjunction with your own project. The addition of Ansible Galaxy access eliminates the need to create git submodules to achieve this result. Given that SCM projects, along with roles or collections, are pulled into and executed from a private job environment, a **<private job directory>** specific to the project within **/tmp** is created by default.

The cache directory is a subdirectory inside the global projects folder. You can copy the content from the cache location to **<job private directory>/requirements_roles**.

By default, automation controller has a system-wide setting that enables you to dynamically download roles from the **roles/requirements.yml** file for SCM projects. You can turn off this setting in the **Job Settings** screen from the navigation panel **Settings → Job**, by switching the **Enable Role Download** toggle to **Off**.

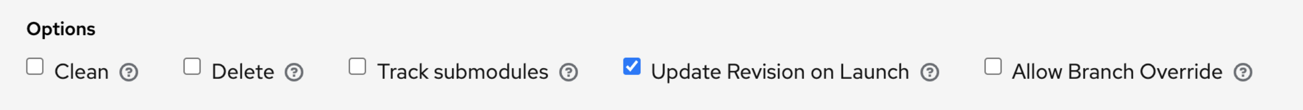
Whenever a project synchronization runs, automation controller determines if the project source and any roles from Galaxy or Collections are out of date with the project. Project updates download the roles inside the update.

If jobs need to pick up a change made to an upstream role, updating the project ensures that this happens. A change to the role means that a new commit was pushed to the *provision-role* source control.

To make this change take effect in a job, you do not have to push a new commit to the *playbooks* repository. You must update the project, which downloads roles to a local cache.

For instance, say you have two git repositories in source control. The first one is *playbooks* and the project in automation controller points to this URL. The second one is *provision-role* and it is referenced by the **roles/requirements.yml** file inside of the *playbooks* git repository.

Jobs download the most recent roles before every job run. Roles and collections are locally cached for performance reasons. You must select **Update Revision on Launch**in the project **Options** to ensure that the upstream role is re-downloaded before each job run:



Options

□ Clean □ Delete □ Track submodules ☑ Update Revision on Launch □ Allow Branch Override

The update happens much earlier in the process than the sync, so this identifies errors and details faster and in a more logical location.

For more information and examples on the syntax of the **requirements.yml** file, see the role requirements section in the Ansible documentation.

If there are any directories that must be specifically exposed, you can specify those in the **Job Settings** screen from the navigation panel **Settings → Job**, in **Paths to Expose to isolated Jobs** You can also update the following entry in the settings file:

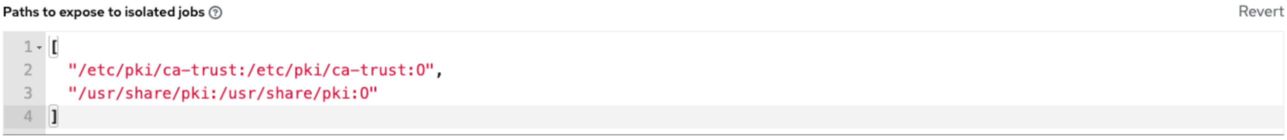
```
AWX_ISOLATION_SHOW_PATHS = ['/list/of/', '/paths']
```

NOTE

If your playbooks need to use keys or settings defined in **AWX_ISOLATION_SHOW_PATHS**, you must add **AWX_ISOLATION_SHOW_PATHS** to **/var/lib/awx/.ssh**.

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

In the UI, you can configure these settings in the **Jobs Settings** window.



```
[
  "/etc/pki/ca-trust:/etc/pki/ca-trust:0",
  "/usr/share/pki:/usr/share/pki:0"
]
```

11.5. COLLECTIONS SUPPORT

Automation controller supports project-specific Ansible collections in job runs. If you specify a collections requirements file in the SCM at **collections/requirements.yml**, automation controller installs collections in that file in the implicit project synchronization before a job run.

Automation controller has a system-wide setting that enables collections to be dynamically downloaded from the **collections/requirements.yml** file for SCM projects. You can turn off this setting in the **Job Settings** screen from the navigation panel **Settings → Job**, by switching the **Enable Collection(s) Download** toggle button to **Off**.

Roles and collections are locally cached for performance reasons, and you select **Update Revision on Launch** in the project **Options** to ensure this:

NOTE

If you also have collections installed in your execution environment, the collections specified in the project's **requirements.yml** file will take precedence when running a job. This precedence applies regardless of the version of the collection. For example, if the collection specified in **requirements.yml** is older than the collection within the execution environment, the collection specified in **requirements.yml** is used.

11.5.1. Using collections with automation hub

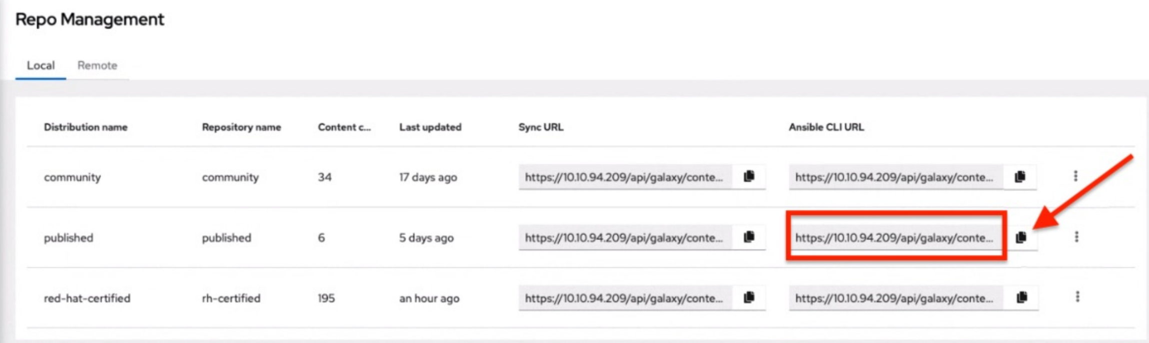
Before automation controller can use automation hub as the default source for collections content, you must create an API token in the automation hub UI. You then specify this token in automation controller.

Use the following procedure to connect to private automation hub or automation hub, the only difference is which URL you specify.

Procedure

1. Go to https://console.redhat.com/ansible/automation-hub/token.
2. Click **Load token**.
3. Click the copy icon to copy the API token to the clipboard.
4. Create a credential by choosing one of the following options:
   a. To use automation hub, create an automation hub credential by using the copied token and pointing to the URLs shown in the **Server URL** and **SSO URL** fields of the token page:
      - Galaxy Server URL = **https://console.redhat.com/ansible/automation-hub/token**

   b. To use private automation hub, create an automation hub credential using a token retrieved from the **Repo Management** dashboard of your private automation hub and pointing to the published repository URL as shown:



Repo Management

Local Remote

Distribution name	Repository name	Content c...	Last updated	Sync URL	Ansible CLI URL
community	community	34	17 days ago	https://10.10.94.209/api/galaxy/conte...	https://10.10.94.209/api/galaxy/conte...
published	published	6	5 days ago	https://10.10.94.209/api/galaxy/conte...	https://10.10.94.209/api/galaxy/conte...
red-hat-certified	rh-certified	195	an hour ago	https://10.10.94.209/api/galaxy/conte...	https://10.10.94.209/api/galaxy/conte...

   You can create different repositories with different namespaces or collections in them. For each repository in automation hub you must create a different credential.

   c. Copy the **Ansible CLI URL** from the UI in the format of **/https://$<hub_url>/api/galaxy/content/<repo you want to pull from>** into the **Galaxy Server URL** field of **Create Credential**:
   For UI specific instructions, see Red Hat Certified, validated, and Ansible Galaxy content in automation hub.

5. Go to the organization for which you want to synchronize content from and add the new credential to the organization. This enables you to associate each organization with the credential, or repository, that you want to use content from.

   Example

   You have two repositories:

   - *Prod*: **Namespace 1** and **Namespace 2**, each with collection **A** and **B** so: **namespace1.collectionA:v2.0.0** and **namespace2.collectionB:v2.0.0**
   - *Stage*: **Namespace 1** with only collection **A** so: **namespace1.collectionA:v1.5.0** on , you have a repository URL for *Prod* and *Stage*.
     You can create a credential for each one.

     Then you can assign different levels of access to different organizations. For example, you can create a **Developers** organization that has access to both repository, while an Operations organization just has access to the **Prod** repository only.

     For UI specific instructions, see Configuring user access for container repositories in private automation hub.

6. If automation hub has self-signed certificates, use the toggle to enable the setting **Ignore Ansible Galaxy SSL Certificate Verification** in **Job Settings**. For automation hub, which uses a signed certificate, use the toggle to disable it instead. This is a global setting:

7. Create a project, where the source repository specifies the necessary collections in a requirements file located in the **collections/requirements.yml** file. For information about the syntax to use, see Using Ansible collections in the Ansible documentation.

8. In the **Projects** list view, click the sync 🔄 icon to update this project. Automation controller fetches the Galaxy collections from the **collections/requirements.yml** file and reports it as changed. The collections are installed for any job template using this project.

NOTE

If updates are required 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 large project (around 10 GB), disk space on **/tmp** may be an issue.

Additional resources

For more information about collections, see Using Ansible Collections.

For more information about how Red Hat publishes one of these official collections, which can be used to automate your install directly, see the AWX Ansible Collection documentation.

CHAPTER 12. PROJECT SIGNING AND VERIFICATION

Project signing and verification lets you sign files in your project directory, then verify whether or not that content has changed in any way, or files have been added or removed from the project unexpectedly. To do this, you require a private key for signing and a matching public key for verifying.

For project maintainers, the supported way to sign content is to use the **ansible-sign** utility, using the *command-line interface* (CLI) supplied with it.

The CLI aims to make it easy to use cryptographic technology such as *GNU Privacy Guard* (GPG) to validate that files within a project have not been tampered with in any way. Currently, GPG is the only supported means of signing and validation.

Automation controller is used to verify the signed content. After a matching public key has been associated with the signed project, automation controller verifies that the files included during signing have not changed, and that files have been added or removed unexpectedly. If the signature is not valid or a file has changed, the project fails to update, and jobs making use of the project will not launch. Verification status of the project ensures that only secure, untampered content can be run in jobs.

If the repository has already been configured for signing and verification, the usual workflow for altering the project becomes the following:

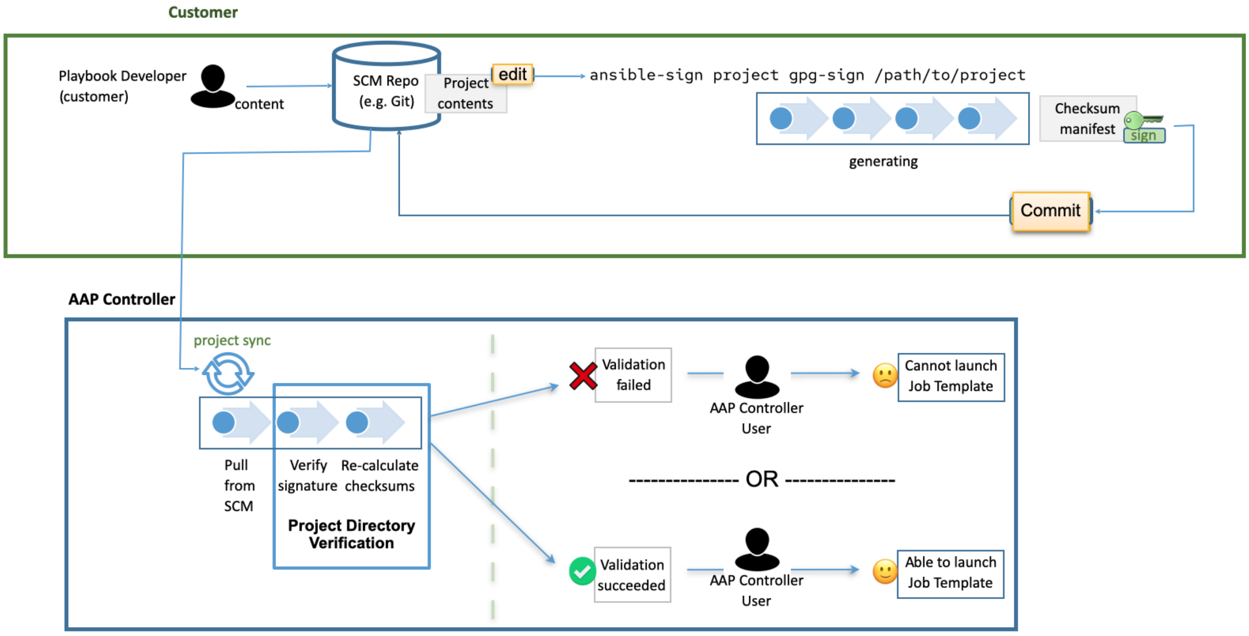
1. You have a project repository set up already and want to make a change to a file.
2. You make the change, and run the following command:

   ```
   ansible-sign project gpg-sign /path/to/project
   ```

   This command updates a checksum manifest and signs it.
3. You commit the change, the updated checksum manifest, and the signature to the repository.

When you synchronize the project, automation controller pulls in the new changes, checks that the public key associated with the project in automation controller matches the private key that the checksum manifest was signed with (this prevents tampering with the checksum manifest itself), then re-calculates the checksums of each file in the manifest to ensure that the checksum matches (and thus that no file has changed). It also ensures that all files are accounted for:

Files must be included in, or excluded from, the **MANIFEST.in** file. For more information on this file, see Sign a project If files have been added or removed unexpectedly, verification fails.



12.1. PREREQUISITES

- RHEL nodes must properly be subscribed to:
  - RHEL subscription with **baseos** and **appstream** repositories must be enabled.
  - Your Red Hat Ansible Automation Platform subscription and the proper channel must be enabled:

    ```
    ansible-automation-platform-2.4-for-rhel-8-x86_64-rpms for RHEL 8
    ansible-automation-platform-2.4-for-rhel-9-x86_64-rpms for RHEL 9
    ```

- A valid GPG public or private keypair is required for signing content. For more information, see How to create GPG keypairs .
  For more information about GPG keys, see the GnuPG documentation.

  Verify that you have a valid GPG keypair in your default GnuPG keyring, with the following command:

  ```
  gpg --list-secret-keys
  ```

  If this command produces no output, or one line of output that states, **trustdb was created**, then you do not have a secret key in your default keyring. In this case, refer to How to create GPG keypairs to learn how to create a new keypair before proceeding. If it produces any other output, you have a valid secret key and are ready to use **ansible-sign**.

12.2. ADDING A GPG KEY TO AUTOMATION CONTROLLER

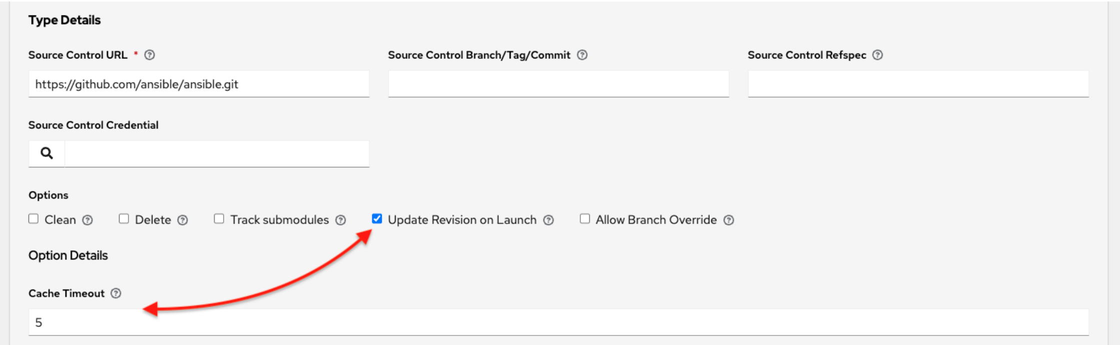
To use the GPG key for content signing and validation in automation controller, add it by running the following command in the CLI:

```
$ gpg --list-keys
$ gpg --export --armour <key fingerprint> > my_public_key.asc
```

1. From the navigation panel, select **Automation Execution → Infrastructure → Credentials**.
2. Click **Create credential**.
3. Give a meaningful name for the new credential, for example, "Infrastructure team public GPG key".
4. In the **Credential type** field, select **GPG Public Key**.
5. Click **Browse** to locate and select the public key file, for example, **my_public_key.asc**.
6. Click **Create credential**.
   You can select this credential in **projects <ug_projects_add>**, and content verification automatically takes place on future project synchronizations.

NOTE

Use the project cache SCM timeout to control how often you want automation controller to re-validate the signed content. When a project is configured to update on launch (of any job template configured to use that project), you can enable the cache timeout setting, which sets it to update after **N** seconds have passed since the last update. If validation is running too often, you can slow down how often project updates occur by specifying the time in the **Cache Timeout** field of the **Options Details** view of the project.



Type Details

Source Control URL *: https://github.com/ansible/ansible.git

Source Control Branch/Tag/Commit:

Source Control Refspec:

Source Control Credential:

Options: □ Clean □ Delete □ Track submodules ☑ Update Revision on Launch □ Allow Branch Override

Option Details

Cache Timeout: 5

12.3. INSTALLING THE ANSIBLE-SIGN CLI UTILITY

Use the **ansible-sign** utility to provide options for the user to sign and verify whether the project is signed.

Procedure

1. Run the following command to install **ansible-sign**:

   ```
   $ dnf install ansible-sign
   ```

2. Verify that **ansible-sign** was successfully installed using the following command:

   ```
   $ ansible-sign --version
   ```

   Output similar to the following indicates that you have successfully installed **ansible-sign**:

   ```
   ansible-sign 0.1
   ```

12.4. SIGN A PROJECT

Signing a project involves an Ansible project directory. For more information on project directory structures, see Sample Ansible setup in the Ansible documentation.

The following sample project has a very simple structure: an inventory file, and two small playbooks under a playbooks directory:

```
$ cd sample-project/
$ tree -a .
.
├── inventory
└── playbooks
    └── get_uptime.yml
    └── hello.yml

  1 directory, 3 files
```

NOTE

The commands used assume that your working directory is the root of your project. **ansible-sign project** commands take the project root directory as their last argument.

Use **.** to indicate the current working directory.

ansible-sign protects content from tampering by taking checksums (SHA256) of all of the secured files in the project, compiling those into a checksum manifest file, and then signing that manifest file.

To sign content, create a **MANIFEST.in** file in the project root directory that tells **ansible-sign** which files to protect.

Internally, **ansible-sign** uses the **distlib.manifest** module of Python's distlib library, therefore **MANIFEST.in** must follow the syntax that this library specifies. For an explanation of the **MANIFEST.in** file directives, see the Python Packaging User Guide.

In the sample project, two directives are included, resulting in the following **MANIFEST.in** file:

```
include inventory
recursive-include playbooks *.yml
```

With this file in place, generate your checksum manifest file and sign it. Both of these steps are achieved in a single **ansible-sign** command:

```
$ ansible-sign project gpg-sign .
```

Successful execution displays output similar to the following:

```
[OK   ] GPG signing successful!
[NOTE ] Checksum manifest: ./.ansible-sign/sha256sum.txt
[NOTE ] GPG summary: signature created
```

The project has now been signed.

Note that the **gpg-sign** subcommand resides under the **project** subcommand.

For signing project content, every command starts with **ansible-sign project**.

Every **ansible-sign project** command takes the project root directory **.** as its final argument.

ansible-sign makes use of your default keyring and looks for the first available secret key that it can find, to sign your project. You can specify a specific secret key to use with the **--fingerprint** option, or even a completely independent GPG home directory with the **--gnupg-home** option.

NOTE

If you are using a desktop environment, GnuPG automatically prompts you for your secret key's passphrase.

If this functionality does not work, or you are working without a desktop environment, for example, through SSH, you can use the **-p --prompt-passphrase** flag after **gpg-sign**, which causes **ansible-sign** to prompt for the password instead.

Note that an **.ansible-sign** directory was created in the project directory. This directory contains the checksum manifest and a detached GPG signature for it.

```
$ tree -a .
.
├── .ansible-sign
│   ├── sha256sum.txt
│   └── sha256sum.txt.sig
├── inventory
├── MANIFEST.in
└── playbooks
    ├── get_uptime.yml
    └── hello.yml
```

12.5. VERIFY YOUR PROJECT

To verify that a signed Ansible project has not been altered, you can use **ansible-sign** to check whether the signature is valid and that the checksums of the files match what the checksum manifest says they should be. The **ansible-sign project gpg-verify** command can be used to automatically verify both of these conditions.

```
$ ansible-sign project gpg-verify .
[OK   ] GPG signature verification succeeded.
[OK   ] Checksum validation succeeded.
```

NOTE

By default, **ansible-sign** makes use of your default GPG keyring to look for a matching public key. You can specify a keyring file with the **--keyring** option, or a different GPG home with the **--gnugpg-home** option.

If verification fails for any reason, information is displayed to help you debug the cause. More verbosity can be enabled by passing the global **--debug** flag, immediately after **ansible-sign** in your commands.

NOTE

When a GPG credential is used in a project, content verification automatically takes place on future project synchronizations.

12.6. AUTOMATE SIGNING

In environments with highly-trusted *Continuous Integration* (CI) environments such as OpenShift or Jenkins, it is possible to automate the signing process.

For example, you can store your GPG private key in a CI platform of choice as a secret, and import that into GnuPG in the CI environment. You can then run through the signing workflow within the normal CI environment.

When signing a project using GPG, the environment variable **ANSIBLE_SIGN_GPG_PASSPHRASE** can be set to the passphrase of the signing key. This can be injected and masked or secured in a CI pipeline.

Depending on the scenario, **ansible-sign** returns with a different exit-code, during both signing and verification. This can also be useful in the context of CI and automation, as a CI environment can act differently based on the failure. For example, it can send alerts for some errors, but fail silently for others.

These are the current exit codes used in **ansible-sign**, which can be considered stable:

Exit code	Approximate meaning	Example scenarios
0	Success	● Signing was successful ● Verification was successful
1	General failure	● The checksum manifest file contained a syntax error during verification ● The signature file did not exist during verification ● **MANIFEST.in** did not exist during signing
2	Checksum verification failure	● The checksum hashes calculated during verification differed from what was in the signed checksum manifest, for example, a project file was changed but the signing process was not re-completed.

Exit code	Approximate meaning	Example scenarios
3	Signature verification failure	● The signer’s public key was not in the user’s GPG keyring ● The wrong GnuPG home directory or keyring file was specified ● The signed checksum manifest file was modified in some way
4	Signing process failure	● The signer’s private key was not found in the GPG keyring ● The wrong GnuPG home directory or keyring file was specified

CHAPTER 13. TOPOLOGY VIEW

Use the **Topology View** to view node type, node health, and specific details about each node if you already have a mesh topology deployed.

To access the topology viewer from the automation controller UI, you must have **System Administrator** permissions.

For more information about automation mesh on a VM-based installation, see the Automation mesh for VM environments.

For more information about automation mesh on an operator-based installation, see the Automation mesh for managed cloud or operator environments.

13.1. ACCESSING THE TOPOLOGY VIEWER

Use the following procedure to access the topology viewer from the automation controller UI.

Procedure

1. From the navigation panel, select **Automation Execution → Infrastructure → Topology View**. The **Topology View** opens and displays a graphical representation of how each receptor node links together.

2. To adjust the zoom levels, or manipulate the graphic views, use the control icons: zoom-in (⊕), zoom-out (⊖), expand (⤧), and reset (⛶) on the toolbar.
   You can also click and drag to pan around; and scroll using your mouse or trackpad to zoom. The fit-to-screen feature automatically scales the graphic to fit on the screen and repositions it in the center. It is particularly useful when you want to see a large mesh in its entirety.

   To reset the view to its default view, click the **Reset view** (⛶) icon.

3. Refer to the **Legend** to identify the type of nodes that are represented.
   For VM-based installations, see Control and execution planes.

   For operator-based installations, see Control and execution planes for more information about each type of node.

   The Legend shows the **node status <node_statuses>** by color, which is indicative of the health of the node. An **Error** status in the Legend includes the **Unavailable** state (as displayed in the Instances list view) plus any future error conditions encountered in later versions of automation controller.

   The following link statuses are also shown in the Legend:

   - **Established**: This is a link state that indicates a peer connection between nodes that are either ready, unavailable, or disabled.

   - **Adding**: This is a link state indicating a peer connection between nodes that were selected to be added to the mesh topology.

   - **Removing**: This is a link state indicating a peer connection between nodes that were selected to be removed from the topology.

4. Hover over a node and the connectors highlight to show its immediate connected nodes (peers) or click a node to retrieve details about it, such as its hostname, node type, and status.

5. Click the link for instance hostname from the details displayed to be redirected to its **Details** page that provides more information about that node, most notably for information about an **Error** status, as in the following example.
   You can use the **Details** page to remove the instance, run a health check on the instance on an as-needed basis, or unassign jobs from the instance. By default, jobs can be assigned to each node. However, you can disable it to exclude the node from having any jobs running on it.

6. Additional resources For more information about creating new nodes and scaling the mesh, see Managing Capacity with Instances.

CHAPTER 14. INVENTORIES

Red Hat Ansible Automation Platform works against a list of managed nodes or hosts in your infrastructure that are logically organized, using an inventory file. You can use the Red Hat Ansible Automation Platform installer inventory file to specify your installation scenario and describe host deployments to Ansible. By using an inventory file, Ansible can manage a large number of hosts with a single command. Inventories also help you use Ansible more efficiently by reducing the number of command line options you have to specify. Inventories are divided into groups and these groups contain the hosts.

Groups can be sourced manually, by entering host names into automation controller, or from one of its supported cloud providers.

> **NOTE**
>
> If you have a custom dynamic inventory script, or a cloud provider that is not yet supported natively in automation controller, you can also import that into automation controller.
>
> For more information, see Inventory file importing in *Configuring automation execution*.

From the navigation panel, select **Automation Execution → Infrastructure → Inventories**. The **Inventories** window displays a list of the inventories that are currently available.

The **Inventory details** page includes:

- **Name**: The inventory name.
- **Status**
  The statuses are:
  - **Success**: When the inventory source sync completed successfully
  - **Disabled**: No inventory source added to the inventory
  - **Error**: When the inventory source sync completed with error
- **Type**: Identifies whether it is a standard inventory, a Smart inventory, or a constructed inventory.
- **Organization**: The organization to which the inventory belongs. The following actions are available for the selected inventory:
  - **Edit** ✎ : Edit the properties for the selected inventory
  - **Copy** ⧉ : Makes a copy of an existing inventory as a template for creating a new one
  - **Delete inventory**: Delete the selected inventory

Click the Inventory name to display the **Details** page for the selected inventory, which shows the inventory’s groups and hosts.

14.1. SMART INVENTORIES

Smart Inventories are collections of hosts defined by a stored search that can be viewed like a standard inventory and can be easily used with job runs. Organization administrators have admin permission for inventories in their organization and can create Smart Inventories.

A Smart Inventory is identified by **KIND=smart**.

You can define a Smart Inventory using the same method being used with Search. **InventorySource** is directly associated with an Inventory.

NOTE

Smart inventories are deprecated and will be removed in a future release. Consider moving to constructed inventories for enhancements and replacement.

The **Inventory** model has the following new fields that are blank by default but are set accordingly for Smart Inventories:

- **kind** is set to **smart** for Smart Inventories.
- **host_filter** is set AND **kind** is set to **smart** for Smart Inventories.

The **host** model has a related endpoint, **smart_inventories** that identifies a set of all the Smart Inventories a host is associated with. The membership table is updated every time a job runs against a smart inventory.

NOTE

To update the memberships more frequently, you can change the **AWX_REBUILD_SMART_MEMBERSHIP** file-based setting to **True**. (The default is False). This updates memberships if the following events occur:

- A new host is added
- An existing host is modified (updated or deleted)
- A new Smart Inventory is added
- An existing Smart Inventory is modified (updated or deleted)

You can view inventories without being editable:

- Names of Host and Group created as a result of an inventory source synchronization.
- Group records cannot be edited or moved.

You cannot create hosts from a Smart Inventory host endpoint (/**inventories**/**N**/**hosts**/) as with a normal inventory. The administrator of a Smart Inventory has permission to edit fields such as the name, description, variables, and the ability to delete, but does not have the permission to modify the **host_filter**, because that affects which hosts (that have a primary membership inside another inventory) are included in the smart inventory.

host_filter only applies to hosts inside of inventories inside the Smart Inventory’s organization.

To modify **host_filter**, you must be the organization administrator of the inventory’s organization. Organization administrators have implicit "admin" access to all inventories inside the organization, therefore, this does not convey any permissions they did not already possess.

Administrators of the Smart Inventory can grant other users (who are not also admins of your organization) permissions such as "use" and "adhoc" to the smart inventory. These permit the actions indicated by the role, as with other standard inventories. However, this does not grant any special permissions to hosts (which live in a different inventory). It does not permit direct read permission to hosts, or permit them to see additional hosts under **/#/hosts/**, although they can still view the hosts under the smart inventory host list.

In some situations, you can modify the following:

- A new Host created manually on Inventory with Inventory sources.
- Groups that were created as a result of inventory source synchronizations.
- Variables on Host and Group are not changeable, even as the local System Administrator.

Hosts associated with the Smart Inventory are manifested at view time. If the results of a Smart Inventory contains more than one host with identical hostnames, only one of the matching hosts is included as part of the Smart Inventory, ordered by Host ID.

14.1.1. Smart Host Filters

You can use a search filter to populate hosts for an inventory. This feature uses the fact searching feature.

Automation controller stores facts generated by an Ansible Playbook during a Job Template in the database whenever **use_fact_cache=True** is set per-Job Template. New facts are merged with existing facts and are per-host. These stored facts can be used to filter hosts with the **/api/v2/hosts** endpoint, using the **GET** query parameter **host_filter**.

For example:

```
/api/v2/hosts?host_filter=ansible_facts__ansible_processor_vcpus=8
```

The **host_filter** parameter permits:

- grouping with ()
- use of the boolean and operator:
  - __ to reference related fields in relational fields
  - __ is used on ansible_facts to separate keys in a JSON key path
  - `[] is used to denote a json array in the path specification
  - **""** can be used in the value when spaces are wanted in the value
- "classic" Django queries may be embedded in the **host_filter**

Examples:

```
/api/v2/hosts/?host_filter=name=localhost
/api/v2/hosts/?host_filter=ansible_facts__ansible_date_time__weekday_number="3"
/api/v2/hosts/?host_filter=ansible_facts__ansible_processor[]="GenuineIntel"
/api/v2/hosts/?host_filter=ansible_facts__ansible_lo__ipv6[]__scope="host"
/api/v2/hosts/?host_filter=ansible_facts__ansible_processor_vcpus=8
```

```
/api/v2/hosts/?host_filter=ansible_facts__ansible_env__PYTHONUNBUFFERED="true"
/api/v2/hosts/?host_filter=(name=localhost or name=database) and (groups__name=east or
groups__name="west coast") and ansible_facts__an
```

You can search **host_filter** by **host name**, **group name**, and **Ansible facts**.

Group search has the following format:

```
groups.name:groupA
```

Fact search has the following format:

```
ansible_facts.ansible_fips:false
```

You can also perform Smart Search searches, which consist of a host name and host description.

```
host_filter=name=my_host
```

NOTE

If a search term in **host_filter** is of string type, to make the value a number (for example, **2.66**) or a JSON keyword (for example, **null**, **true** or **false**) valid, add double quotations around the value to prevent the controller from parsing it as a non-string:

```
host_filter=ansible_facts__packages__dnsmasq[]__version="2.66"
```

14.2. CONSTRUCTED INVENTORIES

You can create a new inventory (called a constructed inventory) from a list of input inventories.

A constructed inventory has copies of hosts and groups in its input inventories, permitting jobs to target groups of servers across many inventories. Groups and hostvars can be added to the inventory content, and hosts can be filtered to limit the size of the constructed inventory.

Constructed inventories use the ansible.builtin.constructed inventory model.

The key factors of a constructed inventory are:

- The normal Ansible hostvars namespace is available
- They provide groups

Constructed inventories take **source_vars** and **limit** as inputs and transform its **input_inventories** into a new inventory, complete with groups. Groups (existing or constructed) can then be referenced in the **limit** field to reduce the number of hosts produced.

You can construct groups based on these host properties:

- RHEL major or minor versions
- Windows hosts
- Cloud based instances tagged in a certain region

- other

The examples described in later sections are organized by the structure of the input inventories.

14.2.1. Filtering on group name and variables

You can filter on a combination of groups and variables. For example, you can filter hosts that match a group variable value and also match a host variable value.

There are two approaches to executing this filter:

- Define two groups: one group to match the group variable and the other group to match the host variable value. Use the **limit** pattern to return the hosts that are in both groups. This is the recommended approach.
- Define one group. In the definition, include the condition that the group and host variables must match specific values. Use the **limit** pattern to return all the hosts in the new group.

Example:

The following inventory file defines four hosts and sets group and host variables. It defines a product group, a sustaining group, and it sets two hosts to a shutdown state.

The goal is to create a filter that returns only production hosts that are shutdown.

```
[account_1234]
host1
host2 state=shutdown

[account_4321]
host3
host4 state=shutdown

[account_1234:vars]
account_alias=product_dev

[account_4321:vars]
account_alias=sustaining
```

The goal here is to return only shutdown hosts that are present in the group with the **account_alias** variable equal to **product_dev**. There are two approaches to this, both shown in YAML format. The first one suggested is recommended.

1. **Construct 2 groups, limit to intersection**
   source_vars:

   ```
   plugin: constructed
   strict: true
   groups:
     is_shutdown: state | default("running") == "shutdown"
     product_dev: account_alias == "product_dev"
   ```

   limit: **is_shutdown:&product_dev**

This constructed inventory input creates a group for both categories and uses the **limit** (host pattern) to only return hosts that are in the intersection of those two groups, which is documented in Patterns:targeting hosts and groups.

When a variable is or is not defined (depending on the host), you can give a default. For example, use **| default("running")** if you know what value it should have when it is not defined. This helps with debugging, as described in Debugging tips.

2. **Construct 1 group, limit to group**

   source_vars:

```
plugin: constructed
strict: true
groups:
  shutdown_in_product_dev: state | default("running") == "shutdown" and account_alias ==
"product_dev"
```

   limit: **shutdown_in_product_dev**

   This input creates one group that only includes hosts that match both criteria. The limit is then just the group name by itself, returning **host2**. The same as the earlier approach.

14.2.2. Debugging tips

It is important to set the **strict** parameter to **true** so that you can debug problems with your templates. If the template fails to render, an error occurs in the associated inventory update for that constructed inventory.

When encountering errors, increase verbosity to get more details.

Giving a default, such as **| default("running")** is a generic use of Jinja2 templates in Ansible. Doing this avoids errors from the template when you set **strict: true**.

You can also set **strict: false**, and so enable the template to produce an error, which results in the host not getting included in that group. However, doing this makes it difficult to debug issues in the future if your templates continue to grow in complexity.

You might still have to debug the intended function of the templates if they are not producing the expected inventory content. For example, if a **groups** group has a complex filter (like **shutdown_in_product_dev**) but does not contain any hosts in the resultant constructed inventory, then use the **compose** parameter to help debug.

For example:

```
source_vars:

plugin: constructed
strict: true
groups:
  shutdown_in_product_dev: state | default("running") == "shutdown" and account_alias ==
"product_dev"
compose:
  resolved_state: state | default("running")
  is_in_product_dev: account_alias == "product_dev"

limit: ``
```

–

Running with a blank **limit** returns all hosts. You can use this to inspect specific variables on specific hosts, giving insight into where problems in the **groups** lie.

14.2.3. Nested groups

A nested group consists of two groups where one is a child of the other. In the following example, the child group has another host inside of it, and the parent group has a variable defined.

Because of the way Ansible core operates, the variable of the parent group is available in the namespace as a playbook is running, and can be used for filtering.

The following example inventory file, **nested.yml** is in YAML format:

```
all:
  children:
    groupA:
      vars:
        filter_var: filter_val
      children:
        groupB:
          hosts:
            host1: {}
    ungrouped:
      hosts:
        host2: {}
```

Because **host1** is in **groupB**, it is also in **groupA**.

Filter on nested group names

Use the following YAML format to filter on nested group names:

```
`source_vars`:

plugin: constructed

`limit`: `groupA`
```

Filter on nested group property

Use the following YAML format to filter on a group variable, even if the host is indirectly a member of that group.

In the inventory content, note that **host2** is not expected to have the variable **filter_var** defined, because it is not in any of the groups. Because **strict: true** is used, use a default value so that hosts without that variable are defined. Using this, **host2**, returns **false** from the expression, instead of producing an error. **host1** inherits the variable from its groups, and is returned.

```
source_vars:

plugin: constructed
strict: true
groups:
  filter_var_is_filter_val: filter_var | default("") == "filter_val"

limit: filter_var_is_filter_val
```

14.2.4. Ansible facts

To create an inventory with Ansible facts, you must run a playbook against the inventory that has the setting **gather_facts: true**. The facts differ system-to-system. The following examples are not intended to address all known scenarios.

14.2.4.1. Filter on environment variables

The following example involves filtering on environmental variables using the YAML format:

```
source_vars:

plugin: constructed
strict: true
groups:
  hosts_using_xterm: ansible_env.TERM == "xterm"

limit: hosts_using_xterm
```

14.2.4.2. Filter hosts by processor type

The following example involves filtering hosts by processor type (Intel) using the YAML format:

```
source_vars:

plugin: constructed
strict: true
groups:
  intel_hosts: "GenuineIntel" in ansible_processor

limit: intel_hosts
```

NOTE

Hosts in constructed inventories are not counted against your license allotment because they are referencing the original inventory host. Additionally, hosts that are disabled in the original inventories are not included in the constructed inventory.

An inventory update run using **ansible-inventory** creates the constructed inventory contents.

This is always configured to update-on-launch before a job, but you can still select a cache timeout value in case this takes too long.

When creating a constructed inventory, the API ensures that it always has one inventory source associated with it. All inventory updates have an associated inventory source, and the fields needed for constructed inventory (**source_vars** and **limit**) are fields already present on the inventory source model.

14.3. INVENTORY PLUGINS

Inventory updates use dynamically-generated YAML files which are parsed by their inventory plugin. In automation controller v4.4, you can provide the inventory plugin configuration directly to automation controller using the inventory source **source_vars** for the following inventory sources:

- Amazon Web Services EC2
- Google Compute Engine
- Microsoft Azure Resource Manager
- VMware vCenter
- Red Hat Satellite 6
- Red Hat Insights
- OpenStack
- Red Hat Virtualization
- Red Hat Ansible Automation Platform
- Terraform State
- OpenShift Virtualization

Newly created configurations for inventory sources contain the default plugin configuration values. If you want your newly created inventory sources to match the output of a legacy source, you must apply a specific set of configuration values for that source. To ensure backward compatibility, automation controller uses "templates" for each of these sources to force the output of inventory plugins into the legacy format.

For more information about sources and their templates, see Supported inventory plugin templates.

source_vars that contain **plugin: foo.bar.baz** as a top-level key are replaced with the fully-qualified inventory plugin name at runtime based on the **InventorySource** source. For example, if you select ec2 for the **InventorySource** then, at run-time, plugin is set to **amazon.aws.aws_ec2**.

14.4. ADD A NEW INVENTORY

Adding a new inventory involves the following components:

- Adding permissions to inventories
- Adding groups to inventories
- Adding a host
- Adding a source
- View completed jobs

Use the following procedure to create a inventory:

Procedure

1. From the navigation panel, select **Automation Execution → Infrastructure → Inventories**. The **Inventories** window displays a list of the inventories that are currently available.

2. Click **Create inventory**, and select the type of inventory to create.

3. Enter the appropriate details into the following fields:

   - **Name**: Enter a name appropriate for this inventory.

   - Optional: **Description**: Enter an arbitrary description as appropriate.

   - **Organization**: Required. Choose among the available organizations.

   - Only applicable to Smart Inventories: **Smart host filter**: Populate the hosts for this inventory by using a search filter.
     Example

     name__icontains=RedHat.

     These options are based on the organization you chose.

     Filters are similar to tags in that tags are used to filter certain hosts that contain those names. Therefore, to populate the **Smart host filter** field, specify a tag that has the hosts you want, not the hosts themselves.

     Filters are case-sensitive.

   - **Instance groups**: Select the instance group or groups for this inventory to run on.
     You can select many instance groups and sort them in the order that you want them run.

   - Optional: **Labels**: Supply labels that describe this inventory, so they can be used to group and filter inventories and jobs.

   - Only applicable to constructed inventories: **Input inventories**: Specify the source inventories to include in this constructed inventory. Empty groups from input inventories are copied into the constructed inventory.

   - Optional:(Only applicable to constructed inventories): **Cached timeout (seconds)**: Set the length of time you want the cache plugin data to timeout.

   - Only applicable to constructed inventories: **Verbosity**: Control the level of output that Ansible produces as the playbook executes related to inventory sources associated with constructed inventories.
     Select the verbosity from:

   - **Normal**

   - **Verbose**

   - **More verbose**

   - **Debug**

   - **Connection Debug**

   - **WinRM Debug**

- ○ **Verbose** logging includes the output of all commands.
  - **More verbose** provides more detail than **Verbose**.
  - **Debug** logging is exceedingly verbose and includes information about SSH operations that can be useful in certain support instances. Most users do not need to see debug mode output.
  - **Connection Debug** enables you to run SSH in verbose mode, providing debugging information about the SSH connection progress.
  - **WinRM Debug** used for verbosity specific to windows remote management
    Click the ❯ icon for information on **How to use the constructed inventory plugin**

- Only applicable to constructed inventories: **Limit**: Restricts the number of returned hosts for the inventory source associated with the constructed inventory. You can paste a group name into the limit field to only include hosts in that group. For more information, see the **Source vars** setting.
- Only applicable to standard inventories: **Options**: Check the **Prevent Instance Group Fallback** option to enable only the instance groups listed in the **Instance Groups** field to execute the job. If unchecked, all available instances in the execution pool are used based on the hierarchy described in Control where a job runs.
- **Variables** (**Source vars** for constructed inventories):
  - **Variables** Variable definitions and values to apply to all hosts in this inventory. Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two.
  - **Source vars** for constructed inventories creates groups, specifically under the **groups** key of the data. It accepts Jinja2 template syntax, renders it for every host, makes a **true** or **false** evaluation, and includes the host in the group (from the key of the entry) if the result is **true**. This is particularly useful because you can paste that group name into the limit field to only include hosts in that group.

4. Click **Create inventory**.

After saving the new inventory, you can proceed with configuring permissions, groups, hosts, sources, and view completed jobs, if applicable to the type of inventory.

14.4.1. Adding permissions to inventories

Use the following procedure to add permissions to inventories:

Procedure

1. From the navigation panel, select **Automation Execution** → **Infrastructure** → **Inventories**.
2. Select a template, and in the **User Access** or **Team Access** tab, click **Add roles**.
3. Select a user or team to add and click **Next**.
4. Select the checkbox next to a name to add one or more users or teams from the list as members.

5. Click **Next**.
6. Select the roles you want the selected users or teams to have. Different resources have different options available.
7. Click **Finish** to apply the roles to the selected users or teams and to add them as members.

The updated roles assigned for each user and team are displayed.

Removing a permission

- To remove roles for a particular user, click the ✖ icon next to its resource.

This launches a confirmation window, asking you to confirm the disassociation.

14.4.2. Adding groups to inventories

Inventories are divided into groups, which can contain hosts and other groups. Groups are only applicable to standard inventories and are not a configurable directly through a Smart Inventory. You can associate an existing group through hosts that are used with standard inventories.

The following actions are available for standard inventories:

- Create a new Group
- Create a new Host
- Run a command on the selected Inventory
- Edit Inventory properties
- View activity streams for Groups and Hosts
- Obtain help building your Inventory

NOTE

Inventory sources are not associated with groups. Spawned groups are top-level and can still have child groups. All of these spawned groups can have hosts.

Procedure

1. From the navigation panel, select **Automation Execution** → **Infrastructure** → **Inventories**.
2. Select the Inventory name you want to add groups to.
3. In the Inventory **Details** page, select the **Groups** tab.
4. Click **Create group**.
5. Enter the appropriate details:
    - **Name**: Required
    - Optional: **Description**: Enter a description as appropriate.

   - Optional: **Variables**: Enter definitions and values to be applied to all hosts in this group. Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two.

6. Click **Create group**.

When you have added a group to a template, the Group **Details** page is displayed.

14.4.2.1. Adding groups within groups

When you have added a group to a template, the Group **Details** page is displayed.

Procedure

1. Select the **Related Groups** tab.
2. Click **Add existing group** to add a group that already exists in your configuration or **Create group** to create a new group.
3. If creating a new group, enter the appropriate details into the required and optional fields:
   - **Name** (required):
   - Optional: **Description**: Enter a description as appropriate.
   - Optional: **Variables**: Enter definitions and values to be applied to all hosts in this group. Enter variables using either JSON or YAML syntax. Use the radio button to toggle between the two.
4. Click **Create group**.
5. The **Create group** window closes and the newly created group is displayed as an entry in the list of groups associated with the group that it was created for.

If you select to add an existing group, available groups appear in a separate selection window.

When you select a group, it is displayed in the list of groups associated with the group.

- To configure additional groups and hosts under the subgroup, click the name of the subgroup from the list of groups and repeat the steps listed in this section.

14.4.2.2. View or edit inventory groups

The groups list view displays all your inventory groups, or you can filter it to only display the root groups. An inventory group is considered a root group if it is not a subset of another group.

You can delete a subgroup without concern for dependencies, because automation controller looks for dependencies such as child groups or hosts. If any exist, a confirmation window displays for you to select whether to delete the root group and all of its subgroups and hosts; or to promote the subgroups so they become the top-level inventory groups, along with their hosts.

14.4.3. Adding hosts to an inventory

You can configure hosts for the inventory and for groups and groups within groups.

Procedure

1. From the navigation panel, select **Automation Execution → Infrastructure → Inventories**.
2. Select the inventory name you want to add groups to.
3. In the Inventory **Details** page, select the **Hosts** tab.
4. Click **Create host**.
5. Select whether to add a host that already exists in your configuration or create a new host.
6. If creating a new host, set the toggle to **On** to include this host while running jobs.
7. Enter the appropriate details:
   - **Name** (required):
   - Optional: **Description**: Enter a description as appropriate.
   - Optional: **Variables**: Enter definitions and values to be applied to all hosts in this group, as in the following example:

     ```
     {
       ansible_user : <username to ssh into>
       ansible_ssh_pass : <password for the username>
       ansible_become_pass: <password for becoming the root>
     }
     ```

     Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two.
8. Click **Create host**.
9. The **Create host** window closes and the newly created host is displayed in the list of hosts associated with the group that it was created for.
   If you select to add an existing host, available hosts appear in a separate selection window.

   When you select a host, it is displayed in the list of hosts associated with the group.
10. You can disassociate a host from this screen by selecting the host and clicking the ✖ icon.

    NOTE

    You can also run ad hoc commands from this screen. For more information, see Running Ad Hoc commands.
11. To configure additional groups for the host, click the name of the host from the list of hosts. This opens the **Details** tab of the selected host.
12. Select the **Groups** tab to configure groups for the host.
13. Click **Associate groups** to associate the host with an existing group. Available groups appear in a separate selection window.
14. Select the groups to associate with the host and click **Confirm**.
    When a group is associated, it is displayed in the list of groups associated with the host.

15. If you used a host to run a job, you can view details about those jobs in the **Completed Jobs** tab of the host.

16. Click **Expanded** to view details about each job.

NOTE

You can create hosts in bulk by using the newly added endpoint in the API, **/api/v2/bulk/host_create**. This endpoint accepts JSON and you can specify the target inventory and a list of hosts to add to the inventory. These hosts must be unique within the inventory. Either all hosts are added, or an error is returned indicating why the operation was not able to complete. Use the **OPTIONS** request to return the relevant schema.

For more information, see Bulk endpoints in the *Automation Controller API Guide*.

14.4.4. Adding a source

Inventory sources are not associated with groups. Spawned groups are top-level and can still have child groups. All of these spawned groups can have hosts. Adding a source to an inventory only applies to standard inventories.

Procedure

1. From the navigation panel, select **Automation Execution** → **Infrastructure** → **Inventories**.

2. Select the inventory name you want to add a source to.

3. In the Inventory **Details** page, select the **Sources** tab.

4. Click **Create source**.

5. Enter the appropriate details:

   - **Name** (required):

   - Optional: **Description**: Enter a description as appropriate.

   - Optional: **Execution Environment**: Click the 🔍 icon or enter the name of the execution environment with which you want to run your inventory imports. For more information on building an execution environment, see Execution environments.

   - **Source**: Choose a source for your inventory. For more information about sources, and supplying the appropriate information, see Inventory sources.

6. When the information for your chosen Inventory sources is complete, you can optionally specify other common parameters, such as verbosity, host filters, and variables.

7. Use the **Verbosity** menu to select the level of output on any inventory source’s update jobs.

8. Use the **Host filter** field to specify only matching host names to be imported into automation controller.

9. In the **Enabled variable** field, specify that automation controller retrieves the enabled state from the dictionary of host variables. You can specify the enabled variable by using dot notation as 'foo.bar', in which case the lookup searches nested dictionaries, equivalent to:

from_dict.get('foo', {}).get('bar', default).

10. If you specified a dictionary of host variables in the **Enabled variable** field, you can give a value to enable on import. For example, for **enabled_var='status.power_state'** and **'enabled_value='powered_on'** in the following host variables, the host is marked **enabled**:

    ```
    {
    "status": {
    "power_state": "powered_on",
    "created": "2020-08-04T18:13:04+00:00",
    "healthy": true
    },
    "name": "foobar",
    "ip_address": "192.168.2.1"
    }
    ```

    If **power_state** is any value other than **powered_on**, then the host is disabled when imported into automation controller. If the key is not found, then the host is enabled.

11. All cloud inventory sources have the following update options:

    - **Overwrite**: If checked, any hosts and groups that were previously present on the external source but are now removed, are removed from the automation controller inventory. Hosts and groups that were not managed by the inventory source are promoted to the next manually created group, or, if there is no manually created group to promote them into, they are left in the "all" default group for the inventory.
      When not checked, local child hosts and groups not found on the external source remain untouched by the inventory update process.

    - **Overwrite variables**: If checked, all variables for child groups and hosts are removed and replaced by those found on the external source.
      When not checked, a merge is performed, combining local variables with those found on the external source.

    - **Update on launch**: Each time a job runs using this inventory, refresh the inventory from the selected source before executing job tasks.
      To avoid job overflows if jobs are spawned faster than the inventory can synchronize, selecting this enables you to configure a **Cache Timeout** to previous cache inventory synchronizations for a certain number of seconds.

      The **Update on launch** setting refers to a dependency system for projects and inventory, and does not specifically exclude two jobs from running at the same time.

      If a cache timeout is specified, then the dependencies for the second job are created, and it uses the project and inventory update that the first job spawned.

      Both jobs then wait for that project or inventory update to finish before proceeding. If they are different job templates, they can then both start and run at the same time, if the system has the capacity to do so. If you intend to use automation controller's provisioning callback feature with a dynamic inventory source, **Update on launch** must be set for the inventory group.

      If you synchronize an inventory source that uses a project that has **Update On launch** set, then the project might automatically update (according to cache timeout rules) before the inventory update starts.

You can create a job template that uses an inventory that sources from the same project that the template uses. In such a case, the project updates and then the inventory updates (if updates are not already in progress, or if the cache timeout has not already expired).

12. Review your entries and selections. This enables you to configure additional details, such as schedules and notifications.

13. To configure schedules associated with this inventory source, click the **Schedules** tab:

    - If schedules are already set up, then review, edit, enable or disable your schedule preferences.

    - If schedules have not been set up, for more information about setting up schedules, see Schedules.

14.4.5. Configuring notifications for the source

Use the following procedure to configure notifications for the source:

Procedure

1. From the navigation panel, select **Automation Execution** → **Infrastructure** → **Inventories**.

2. Select the inventory name you want to configure notifications for.

3. In the inventory **Details** page, select the **Notifications** tab.

   NOTE

   The **Notifications** tab is only present when you have saved the newly-created source.

4. If notifications are already set up, use the toggles to enable or disable the notifications to use with your particular source. For more information, see Enable and Disable Notifications.

5. If you have not set up notifications, see Notifications for more information.

6. Review your entries and selections.

7. Click **Save**.

When you define a source, it is displayed in the list of sources associated with the inventory. From the **Sources** tab you can perform a sync on a single source, or sync all of them at once. You can also perform additional actions such as scheduling a sync process, and edit or delete the source.

14.4.5.1. Inventory sources

Choose a source which matches the inventory type against which a host can be entered:

- Sourcing from a Project

- Amazon Web Services EC2

- Google Compute Engine

- Microsoft Azure Resource Manager

- VMware vCenter
- Red Hat Satellite 6
- Red Hat Insights
- OpenStack
- Red Hat Virtualization
- Red Hat Ansible Automation Platform
- Terraform State

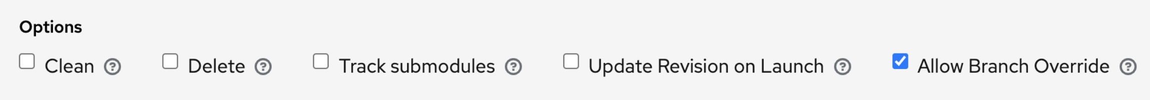
14.4.5.1.1. Sourcing from a Project

An inventory that is sourced from a project means that it uses the SCM type from the project it is tied to. For example, if the project's source is from GitHub, then the inventory uses the same source.

Use the following procedure to configure a project-sourced inventory:

Procedure

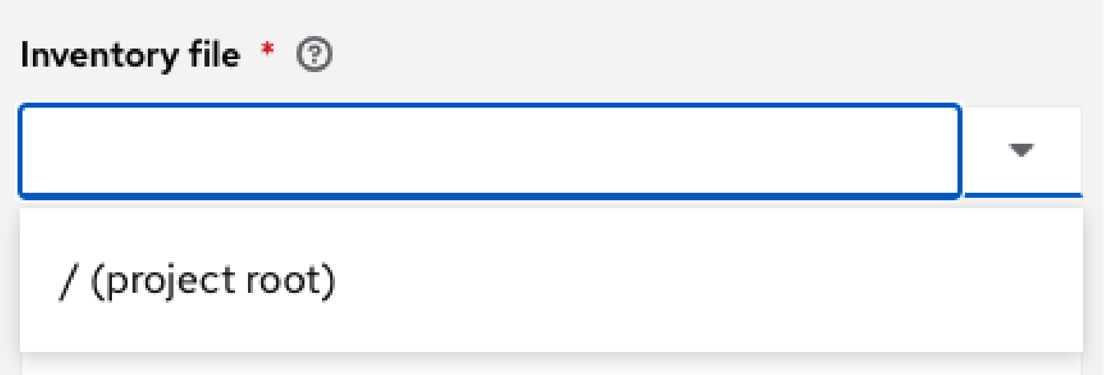
1. From the navigation panel, select **Automation Execution → Infrastructure → Inventories**.
2. Select the inventory name you want a source to and click the **Sources** tab.
3. Click **Create source**.
4. In the **Create source** page, select **Sourced from a Project** from the **Source** list.
5. Enter the following details in the additional fields:
   - Optional: **Source control branch/tag/commit**: Enter the SCM branch, tags, commit hashes, arbitrary refs, or revision number (if applicable) from the source control (Git or Subversion) to checkout.
     This field only displays if the sourced project has the **Allow branch override** option checked. For further information, see SCM Types - Git and Subversion .

     Options

     □ Clean ⓘ □ Delete ⓘ □ Track submodules ⓘ □ Update Revision on Launch ⓘ ☑ Allow Branch Override ⓘ

     Some commit hashes and refs might not be available unless you also give a custom refspec in the next field. If left blank, the default is HEAD which is the last checked out Branch/Tag/Commit for this project.
   - Optional: **Credential**: Specify the credential to use for this source.
   - **Project** (required): Pre-populates with a default project, otherwise, specify the project this inventory is using as its source. Click the 🔍 icon to choose from a list of projects. If the list is extensive, use the search to narrow the options.
   - **Inventory file** (required): Select an inventory file associated with the sourced project. If not already populated, you can type it into the text field within the menu to filter extraneous file types. In addition to a flat file inventory, you can point to a directory or an inventory script.



6. Optional: You can specify the verbosity, host filter, enabled variable/value, and update options as described in Adding a source.

7. Optional: To pass to the custom inventory script, you can set environment variables in the **Source variables** field. You can also place inventory scripts in source control and then run it from a project. For more information, see Inventory File Importing in *Configuring automation execution*.

NOTE

If you are executing a custom inventory script from SCM, ensure that you set the execution bit (**chmod +x**) for the script in your upstream source control.

If you do not, automation controller throws a **[Error 13] Permission denied** error on execution.

14.4.5.1.2. Amazon Web Services EC2

Use the following procedure to configure an AWS EC2-sourced inventory.

Procedure

1. From the navigation panel, select **Automation Execution** → **Infrastructure** → **Inventories**.

2. Select the inventory name you want a source to and click the **Sources** tab.

3. Click **Create source**.

4. In the **Create source** page, select **Amazon EC2** from the **Source** list.

5. The **Create source** window expands with additional fields. Enter the following details:

   - Optional: **Credential**: Choose from an existing AWS credential. For more information, see Managing user credentials.
     If automation controller is running on an EC2 instance with an assigned IAM Role, the credential can be omitted, and the security credentials from the instance metadata are used instead. For more information about using IAM Roles, see IAM roles for Amazon EC2_documentation_at_Amazon documentation at Amazon.

6. Optional: You can specify the verbosity, host filter, enabled variables or values, and update options as described in Adding a source.

7. Use the **Source variables** field to override variables used by the **aws_ec2** inventory plugin. Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two. For more information about these variables, see the aws inventory plugin

documentation.

NOTE

If you only use **include_filters**, the AWS plugin always returns all the hosts. To use this correctly, the first condition on the **or** must be on **filters** and then build the rest of the **OR** conditions on a list of **include_filters**.

14.4.5.1.3. Google Compute Engine

Use the following procedure to configure a Google-sourced inventory:

Procedure

1. From the navigation panel, select **Automation Execution** → **Infrastructure** → **Inventories**.
2. Select the inventory name you want a source to and click the **Sources** tab.
3. Click **Create source**.
4. In the **Add new source** page, select **Google Compute Engine** from the **Source** list.
5. The **Create source** window expands with the required **Credential** field. Choose from an existing GCE Credential. For more information, see [Credentials].
6. Optional: You can specify the verbosity, host filter, enabled variables or values, and update options as described in Adding a source.
7. Use the **Source Variables** field to override variables used by the **gcp_compute** inventory plugin. Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two. For more information about these variables, see the gcp_compute inventory plugin documentation.

14.4.5.1.4. Microsoft Azure resource manager

Use the following procedure to configure an Microsoft Azure Resource Manager-sourced inventory.

Procedure

1. From the navigation panel, select **Automation Execution** → **Infrastructure** → **Inventories**.
2. Select the inventory name you want a source to and click the **Sources** tab.
3. Click **Create source**.
4. In the **Create source** page, select **Microsoft Azure Resource Manager** from the **Source** list.
5. Enter the following details in the additional fields:
6. Optional: **Credential**: Choose from an existing Azure Credential. For more information, see Managing user credentials..
7. Optional: You can specify the verbosity, host filter, enabled variables or values, and update options as described in Adding a source.
8. Use the **Source variables** field to override variables used by the **azure_rm** inventory plugin.

Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two. For more information about these variables, see the azure_rm inventory plugin documentation.

14.4.5.1.5. VMware vCenter

Use the following procedure to configure a VMWare-sourced inventory.

Procedure

1. From the navigation panel, select **Automation Execution → Infrastructure → Inventories**.
2. Select the inventory name you want a source to and click the **Sources** tab.
3. Click **Create source**.
4. In the **Create source** page, select **VMware vCenter** from the **Source** list.
5. The **Create source** window expands with additional fields. Enter the following details:
   - Optional: **Credential**: Choose from an existing VMware credential. For more information, see Managing user credentials.
6. Optional: You can specify the verbosity, host filter, enabled variables or values, and update options as described in Adding a source.
7. Use the **Source Variables** field to override variables used by the **vmware_inventory** inventory plugin. Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two. For more information about these variables, see the vmware_inventory inventory plugin.

NOTE

VMWare properties have changed from lower case to camel case. Automation controller provides aliases for the top-level keys, but lower case keys in nested properties have been discontinued. For a list of valid and supported properties, see Using Virtual machine attributes in VMware dynamic inventory plugin.

14.4.5.1.6. Red Hat Satellite 6

Use the following procedure to configure a Red Hat Satellite-sourced inventory.

Procedure

1. From the navigation panel, select **Automation Execution → Infrastructure → Inventories**.
2. Select the inventory name you want a source to and click the **Sources** tab.
3. Click **Create source**.
4. In the **Create source** page,, select **Red Hat Satellite 6** from the **Source** list.
5. The **Create source** window expands with additional fields. Enter the following details:
   - Optional: **Credential**: Choose from an existing Satellite Credential. For more information, see Managing user credentials.

6. Optional: You can specify the verbosity, host filter, enabled variables or values, and update options as described in Adding a source.

7. Use the **Source Variables** field to specify parameters used by the **foreman** inventory source. Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two. For more information about these variables, see the Foreman inventory source in the Ansible documentation.

If you meet an issue with the automation controller inventory not having the "related groups" from Satellite, you might need to define these variables in the inventory source. For more information, see Red Hat Satellite 6 .

If you see the message, **"no foreman.id" variable(s) when syncing the inventory**, see the solution on the Red Hat Customer Portal at: https://access.redhat.com/solutions/5826451. Be sure to login with your customer credentials to access the full article.

14.4.5.1.7. Red Hat Insights

Use the following procedure to configure a Red Hat Insights-sourced inventory.

Procedure

1. From the navigation panel, select **Automation Execution → Infrastructure → Inventories**.

2. Select the inventory name you want a source to and click the **Sources** tab.

3. Click **Create source**.

4. In the **Create source** page, select **Red Hat Insights** from the **Source** list.

5. The **Create source** window expands with additional fields. Enter the following details:

   - Optional: **Credential**: Choose from an existing Red Hat Insights Credential. For more information, see Managing user credentials.

6. Optional: You can specify the verbosity, host filter, enabled variables or values, and update options as described in Adding a source.

7. Use the **Source Variables** field to override variables used by the **insights** inventory plugin. Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two. For more information about these variables, see insights inventory plugin.

14.4.5.1.8. OpenStack

Use the following procedure to configure an OpenStack-sourced inventory.

Procedure

1. From the navigation panel, select **Automation Execution → Infrastructure → Inventories**.

2. Select the inventory name you want a source to and click the **Sources** tab.

3. Click **Create source**.

4. In the **Create source** page, select **OpenStack** from the **Source** list.

5. The **Create source** window expands with additional fields. Enter the following details:
   - Optional: **Credential**: Choose from an existing OpenStack Credential. For more information, see Managing user credentials.
6. Optional: You can specify the verbosity, host filter, enabled variables or values, and update options as described in Adding a source.
7. Use the **Source Variables** field to override variables used by the **openstack** inventory plugin. Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two. For more information about these variables, see openstack inventory plugin.

14.4.5.1.9. Red Hat Virtualization

Use the following procedure to configure a Red Hat virtualization-sourced inventory.

Procedure

1. From the navigation panel, select **Automation Execution → Infrastructure → Inventories**.
2. Select the inventory name you want a source to and click the **Sources** tab.
3. Click **Create source**.
4. In the **Create source** page, select **Red Hat Virtualization** from the **Source** list.
5. The **Create source** window expands with additional fields. Enter the following details:
   - Optional: **Credential**: Choose from an existing Red Hat Virtualization Credential. For more information, see Managing user credentials.
6. Optional: You can specify the verbosity, host filter, enabled variables or values, and update options as described in Adding a source.
7. Use the **Source Variables** field to override variables used by the **ovirt** inventory plugin. Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two. For more information about these variables, see ovirt inventory plugin

> **NOTE**
>
> Red Hat Virtualization (ovirt) inventory source requests are secure by default. To change this default setting, set the key **ovirt_insecure** to **true** in **source_variables**, which is only available from the API details of the inventory source at the **/api/v2/inventory_sources/N/** endpoint.

14.4.5.1.10. Red Hat Ansible Automation Platform

Use the following procedure to configure an automation controller-sourced inventory.

Procedure

1. From the navigation panel, select **Automation Execution → Infrastructure → Inventories**.
2. Select the inventory name you want a source to and click the **Sources** tab.
3. Click **Create source**.

4. In the **Create source** page, select **Red Hat Ansible Automation Platform** from the **Source** list.

5. The **Create source** window expands with additional fields. Enter the following details:

   - Optional: **Credential**: Choose from an existing Red Hat Ansible Automation Platform Credential. For more information, see Managing user credentials.

6. Optional: You can specify the verbosity, host filter, enabled variables or values, and update options as described in Adding a source.

7. Use the **Source Variables** field to override variables used by the **controller** inventory plugin. Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two. For more information about these variables, see Controller inventory plugin. This requires your Red Hat Customer login.

14.4.5.1.11. Terraform State

This inventory source uses the terraform_state inventory plugin from the cloud.terraform collection. The plugin parses a terraform state file and add hosts for AWS EC2, GCE, and Microsoft Azure instances.

Procedure

1. From the navigation panel, select **Automation Execution → Projects**.

2. On the **Projects** page, click **Create project** to start the **Create Project** window.

   - Enter the appropriate details according to the steps in Adding a new project.

3. From the navigational panel, select **Automation Execution → Infrastructure → Inventories**.

4. Select the inventory that you want to add a source to.

5. In the **Sources** tab, click **Create source**.

6. From the **Source** menu, select **Terraform State**.

   - The **Create source** window expands with the optional **Credential** field.
     Choose an existing Terraform backend configuration credential. For more information, see Terraform backend configuration.

7. Enable the options to **Overwrite** and **Update on Launch**.

8. Use the **Source Variables** field to override variables used by the **terraform_state** inventory plugin. Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two. For more information about these variables, see the terraform_state file.
   The **backend_type** variable is required by the Terraform State inventory plugin. This should match the remote backend configured in the **Terraform backend credential**. The following is an example Amazon S3 backend:

   ```
   backend_type: s3
   ```

9. Select an **Execution Environment** that has a Terraform binary. This is required for the inventory plugin to run the Terraform commands that read inventory data from the Terraform state file.

Additional resources

For more information, see the Terraform EE readme that has an example execution environment configuration with a Terraform binary.

14.4.5.1.11.1. Terraform provider for Ansible Automation Platform

Inventories created this way are managed by Terraform and you must not edit them in Ansible Automation Platform as it can introduce drift to the Terraform deployment.

You can create inventories and hosts within the Terraform configuration by using the Terraform provider for Ansible Automation Platform. For more information, see the AAP Provider section of the Terraform documentation.

14.4.5.1.12. OpenShift Virtualization

This inventory source uses a cluster that is able to deploy Red Hat OpenShift Container Platform Virtualization. To configure a Red Hat OpenShift Container Platform Virtualization, you need a virtual machine deployed in a specific namespace and an OpenShift or Kubernetes API Bearer Token credential.

Procedure

1. From the navigational panel, select **Automation Execution → Infrastructure → Inventories**.
2. Select the inventory that you want to add a source to.
3. In the **Sources** tab, click **Add source**.
4. From the **Source** menu, select **OpenShift Virtualization**.
   - The **Add new source** window expands with the required **Credential** field.
     Choose from an existing Kubernetes API Bearer Token credential. For more information, see OpenShift or Kubernetes API Bearer Token credential type . In this example, the **cmv2.engineering.redhat.com** credential is used.
5. You can optionally specify the **Verbosity**, **Host Filter**, **Enabled Variable/Value**, and **Update options** as described in the Adding a source steps.
6. Use the **Source Variables** field to override variables used by the **kubernetes** inventory plugin. Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two. For more information about these variables, see the kubevirt.core.kubevirt inventory source documentation.
   In the following example, the connections variable is used to specify access to a particular namespace in a cluster:

   ```
   ---
   connections:
   - namespaces:
     - hao-test
   ```

7. Click **Save** and then click **Sync** to sync the inventory.

14.4.5.2. Export old inventory scripts

Despite the removal of the custom inventory scripts API, the scripts are still saved in the database. The commands described in this section enable you to recover the scripts from the database in a format that is suitable for you to subsequently check into source control.

Use the following commands:

```
$ awx-manage export_custom_scripts --filename=my_scripts.tar

Dump of old custom inventory scripts at my_scripts.tar
```

Making use of the output:

```
$ mkdir my_scripts
$ tar -xf my_scripts.tar -C my_scripts
```

The name of the scripts has the form: ***<pk>*_<name>**. This is the naming scheme used for project folders.

```
$ ls my_scripts
10inventory_script_rawhook _19 _30inventory_script_listenhospital _11inventory_script_upperorder
_1inventory_script_commercialinternet45 _4inventory_script_whitestring
_12inventory_script_eastplant _22inventory_script_pinexchange
_5inventory_script_literaturepossession _13inventory_script_governmentculture
_23inventory_script_brainluck _6inventory_script_opportunitytelephone
_14inventory_script_bottomguess _25inventory_script_buyerleague _7inventory_script_letjury
_15inventory_script_wallisland _26inventory_script_lifesport _8random_inventory_script
16inventory_script_wallisland _27inventory_script_exchangesomewhere _9random_inventory_script
_17inventory_script_bidstory            _28inventory_script_boxchild _18p
_29__inventory_script_wearstress
```

Each file contains a script. Scripts can be **bash**/**python**/**ruby**/**more**, so the extension is not included. They are all directly executable. Executing the script dumps the inventory data.

```
$ ./my_scripts/11__inventory_script_upperorder
{"group\ud801\udcb0\uc20e\u7b0e\ud81c\udfeb\ub12b\ub4d0\u9ac6\ud81e\udf07\u6ff9\uc17b":
{"hosts":
["host_\ud821\udcad\u68b6\u7a51\u93b4\u69cf\uc3c2\ud81f\uddbe\ud820\udc92\u3143\u62c7",
"host_\u6057\u3985\u1f60\ufefb\u1b22\ubd2d\ua90c\ud81a\udc69\u1344\u9d15",
"host_\u78a0\ud820\udef3\u925e\u69da\ua549\ud80c\ude7e\ud81e\udc91\ud808\uddd1\u57d6\ud801\
ude57",
"host_\ud83a\udc2d\ud7f7\ua18a\u779a\ud800\udf8b\u7903\ud820\udead\u4154\ud808\ude15\u9711",

"host_\u18a1\u9d6f\u08ac\u74c2\u54e2\u740e\u5f02\ud81d\uddee\ufbd6\u4506"], "vars":
{"ansible_host": "127.0.0.1", "ansible_connection":
"local"}}}
```

You can verify functionality with **ansible-inventory**. This gives the same data, but reformatted.

```
$ ansible-inventory -i ./my_scripts/_11__inventory_script_upperorder --list --export
```

In the preceding example, you can **cd** into **my_scripts** and then issue a **git init** command, add the scripts you want, push it to source control, and then create an SCM inventory source in the user interface.

For more information about syncing or using custom inventory scripts, see Inventory file importing in *Configuring automation execution*.

14.5. VIEW COMPLETED JOBS

If you use an inventory to run a job, you can view details about those jobs in the **Jobs** tab of the inventory and click **Expanded** to view details about each job.

14.6. RUNNING AD HOC COMMANDS

Ad hoc refers to using Ansible to perform a quick command, using /usr/bin/ansible, rather than the orchestration language, which is /usr/bin/ansible-playbook. An example of an ad hoc command might be rebooting 50 machines in your infrastructure. Anything you can do ad hoc can be accomplished by writing a playbook. Playbooks can also glue many other operations together.

Use the following procedure to run an ad hoc command:

Procedure

1. From the navigation panel, select **Automation Execution → Infrastructure → Inventories**.

2. Select the inventory name you want to run an ad hoc command with.

3. Select an inventory source from the **Hosts** or **Groups** tab. The inventory source can be a single group or host, a selection of many hosts, or a selection of many groups.

4. Click **Run Command**. The Run command window opens.

5. Enter the following information:

    - **Module**: Select one of the modules that the supports running commands against.

command	apt_repository	mount	win_service
shell	apt_rpm	ping	win_updates
yum	service	selinux	win_group
apt	group	setup	win_user
apt_key	user	win_ping	win_user

    - **Arguments**: Provide arguments to be used with the module you selected.

    - **Limit**: Enter the limit used to target hosts in the inventory. To target all hosts in the inventory enter **all** or *, or leave the field blank. This is automatically populated with whatever was selected in the previous view before clicking the launch button.

    - **Machine Credential**: Select the credential to use when accessing the remote hosts to run the command. Choose the credential containing the username and SSH key or password that Ansible needs to log in to the remote hosts.

    - **Verbosity**: Select a verbosity level for the standard output.

- **Forks**: If needed, select the number of parallel or simultaneous processes to use while executing the command.
- **Show Changes**: Select to enable the display of Ansible changes in the standard output. The default is OFF.
- **Enable Privilege Escalation**: If enabled, the playbook is run with administrator privileges. This is the equivalent of passing the **--become** option to the **ansible** command.
- **Extra Variables**: Provide extra command line variables to be applied when running this inventory. Enter variables using either JSON or YAML syntax. Use the radio button to toggle between the two.

6. Click **Next** to select the execution environment you want the ad hoc command to be run against.
7. Click **Next** to select the credential you want to use.
8. Click **Launch**. The results display in the **Output** tab of the module’s job window.

CHAPTER 15. SUPPORTED INVENTORY PLUGIN TEMPLATES

After upgrade to 4.x, existing configurations are migrated to the new format that produces a backwards compatible inventory output. Use the following templates to aid in migrating your inventories to the new style inventory plugin output.

- Amazon Web Services EC2
- Google Compute Engine
- Microsoft Azure Resource Manager
- VMware vCenter
- Red Hat Satellite 6
- OpenStack
- Red Hat Virtualization
- Red Hat Ansible Automation Platform

15.1. AMAZON WEB SERVICES EC2

```
compose:
  ansible_host: public_ip_address
  ec2_account_id: owner_id
  ec2_ami_launch_index: ami_launch_index | string
  ec2_architecture: architecture
  ec2_block_devices: dict(block_device_mappings | map(attribute='device_name') | list |
zip(block_device_mappings | map(attribute='ebs.volume_id') | list))
  ec2_client_token: client_token
  ec2_dns_name: public_dns_name
  ec2_ebs_optimized: ebs_optimized
  ec2_eventsSet: events | default("")
  ec2_group_name: placement.group_name
  ec2_hypervisor: hypervisor
  ec2_id: instance_id
  ec2_image_id: image_id
  ec2_instance_profile: iam_instance_profile | default("")
  ec2_instance_type: instance_type
  ec2_ip_address: public_ip_address
  ec2_kernel: kernel_id | default("")
  ec2_key_name: key_name
  ec2_launch_time: launch_time | regex_replace(" ", "T") | regex_replace("(\+)(\d\d):(\d)(\d)$",
".\g<2>\g<3>Z")
  ec2_monitored: monitoring.state in ['enabled', 'pending']
  ec2_monitoring_state: monitoring.state
  ec2_persistent: persistent | default(false)
  ec2_placement: placement.availability_zone
  ec2_platform: platform | default("")
  ec2_private_dns_name: private_dns_name
  ec2_private_ip_address: private_ip_address
  ec2_public_dns_name: public_dns_name
  ec2_ramdisk: ramdisk_id | default("")
  ec2_reason: state_transition_reason
  ec2_region: placement.region
  ec2_requester_id: requester_id | default("")
  ec2_root_device_name: root_device_name
  ec2_root_device_type: root_device_type
  ec2_security_group_ids: security_groups | map(attribute='group_id') | list |  join(',')
  ec2_security_group_names: security_groups | map(attribute='group_name') | list |  join(',')
  ec2_sourceDestCheck: source_dest_check | default(false) | lower | string
  ec2_spot_instance_request_id: spot_instance_request_id | default("")
  ec2_state: state.name
  ec2_state_code: state.code
  ec2_state_reason: state_reason.message if state_reason is defined else ""
  ec2_subnet_id: subnet_id | default("")
  ec2_tag_Name: tags.Name
  ec2_virtualization_type: virtualization_type
  ec2_vpc_id: vpc_id | default("")
filters:
  instance-state-name:
  - running
groups:
  ec2: true
hostnames:
  - network-interface.addresses.association.public-ip
  - dns-name
  - private-dns-name
keyed_groups:
  - key: image_id | regex_replace("[^A-Za-z0-9\_]", "_")
    parent_group: images
    prefix: ''
    separator: ''
  - key: placement.availability_zone
    parent_group: zones
    prefix: ''
    separator: ''
  - key: ec2_account_id | regex_replace("[^A-Za-z0-9\_]", "_")
    parent_group: accounts
    prefix: ''
    separator: ''
  - key: ec2_state | regex_replace("[^A-Za-z0-9\_]", "_")
    parent_group: instance_states
    prefix: instance_state
  - key: platform | default("undefined") | regex_replace("[^A-Za-z0-9\_]", "_")
    parent_group: platforms
    prefix: platform
  - key: instance_type | regex_replace("[^A-Za-z0-9\_]", "_")
    parent_group: types
    prefix: type
  - key: key_name | regex_replace("[^A-Za-z0-9\_]", "_")
    parent_group: keys
    prefix: key
  - key: placement.region
    parent_group: regions
    prefix: ''
    separator: ''
  - key: security_groups | map(attribute="group_name") | map("regex_replace", "[^A-Za-z0-9\_]", "_") |
list
```

```
    parent_group: security_groups
    prefix: security_group
  - key: dict(tags.keys() | map("regex_replace", "[^A-Za-z0-9\_]", "_") | list | zip(tags.values()
      | map("regex_replace", "[^A-Za-z0-9\_]", "_") | list))
    parent_group: tags
    prefix: tag
  - key: tags.keys() | map("regex_replace", "[^A-Za-z0-9\_]", "_") | list
    parent_group: tags
    prefix: tag
  - key: vpc_id | regex_replace("[^A-Za-z0-9\_]", "_")
    parent_group: vpcs
    prefix: vpc_id
  - key: placement.availability_zone
    parent_group: '{{ placement.region }}'
    prefix: ''
    separator: ''
plugin: amazon.aws.aws_ec2
use_contrib_script_compatible_sanitization: true
```

15.2. GOOGLE COMPUTE ENGINE

```
auth_kind: serviceaccount
compose:
  ansible_ssh_host: networkInterfaces[0].accessConfigs[0].natIP |
default(networkInterfaces[0].networkIP)
  gce_description: description if description else None
  gce_id: id
  gce_image: image
  gce_machine_type: machineType
  gce_metadata: metadata.get("items", []) | items2dict(key_name="key", value_name="value")
  gce_name: name
  gce_network: networkInterfaces[0].network.name
  gce_private_ip: networkInterfaces[0].networkIP
  gce_public_ip: networkInterfaces[0].accessConfigs[0].natIP | default(None)
  gce_status: status
  gce_subnetwork: networkInterfaces[0].subnetwork.name
  gce_tags: tags.get("items", [])
  gce_zone: zone
hostnames:
- name
- public_ip
- private_ip
keyed_groups:
- key: gce_subnetwork
  prefix: network
- key: gce_private_ip
  prefix: ''
  separator: ''
- key: gce_public_ip
  prefix: ''
  separator: ''
- key: machineType
  prefix: ''
  separator: ''
- key: zone
```

```
  prefix: ''
  separator: ''
- key: gce_tags
  prefix: tag
- key: status | lower
  prefix: status
- key: image
  prefix: ''
  separator: ''
plugin: google.cloud.gcp_compute
retrieve_image_info: true
use_contrib_script_compatible_sanitization: true
```

15.3. MICROSOFT AZURE RESOURCE MANAGER

```
conditional_groups:
  azure: true
default_host_filters: []
fail_on_template_errors: false
hostvar_expressions:
  computer_name: name
  private_ip: private_ipv4_addresses[0] if private_ipv4_addresses else None
  provisioning_state: provisioning_state | title
  public_ip: public_ipv4_addresses[0] if public_ipv4_addresses else None
  public_ip_id: public_ip_id if public_ip_id is defined else None
  public_ip_name: public_ip_name if public_ip_name is defined else None
  tags: tags if tags else None
  type: resource_type
keyed_groups:
- key: location
  prefix: ''
  separator: ''
- key: tags.keys() | list if tags else []
  prefix: ''
  separator: ''
- key: security_group
  prefix: ''
  separator: ''
- key: resource_group
  prefix: ''
  separator: ''
- key: os_disk.operating_system_type
  prefix: ''
  separator: ''
- key: dict(tags.keys() | map("regex_replace", "^(.*)$", "\1_") | list | zip(tags.values() | list)) if tags else []
  prefix: ''
  separator: ''
plain_host_names: true
plugin: azure.azcollection.azure_rm
use_contrib_script_compatible_sanitization: true
```

15.4. VMWARE VCENTER

```
compose:
  ansible_host: guest.ipAddress
  ansible_ssh_host: guest.ipAddress
  ansible_uuid: 99999999 | random | to_uuid
  availablefield: availableField
  configissue: configIssue
  configstatus: configStatus
  customvalue: customValue
  effectiverole: effectiveRole
  guestheartbeatstatus: guestHeartbeatStatus
  layoutex: layoutEx
  overallstatus: overallStatus
  parentvapp: parentVApp
  recenttask: recentTask
  resourcepool: resourcePool
  rootsnapshot: rootSnapshot
  triggeredalarmstate: triggeredAlarmState
filters:
- runtime.powerState == "poweredOn"
keyed_groups:
- key: config.guestId
  prefix: ''
  separator: ''
- key: '"templates" if config.template else "guests"'
  prefix: ''
  separator: ''
plugin: community.vmware.vmware_vm_inventory
properties:
- availableField
- configIssue
- configStatus
- customValue
- datastore
- effectiveRole
- guestHeartbeatStatus
- layout
- layoutEx
- name
- network
- overallStatus
- parentVApp
- permission
- recentTask
- resourcePool
- rootSnapshot
- snapshot
- triggeredAlarmState
- value
- capability
- config
- guest
- runtime
- storage
- summary
strict: false
with_nested_properties: true
```

15.5. RED HAT SATELLITE 6

```
group_prefix: foreman_
keyed_groups:
- key: foreman['environment_name'] | lower | regex_replace(' ', '') | regex_replace('[^A-Za-z0-9_]', '_') |
regex_replace('none', '')
  prefix: foreman_environment_
  separator: ''
- key: foreman['location_name'] | lower | regex_replace(' ', '') | regex_replace('[^A-Za-z0-9_]', '_')
  prefix: foreman_location_
  separator: ''
- key: foreman['organization_name'] | lower | regex_replace(' ', '') | regex_replace('[^A-Za-z0-9_]', '_')
  prefix: foreman_organization_
  separator: ''
- key: foreman['content_facet_attributes']['lifecycle_environment_name'] | lower | regex_replace(' ', '') |
regex_replace('[^A-Za-z0-9_]', '_')
  prefix: foreman_lifecycle_environment_
  separator: ''
- key: foreman['content_facet_attributes']['content_view_name'] | lower | regex_replace(' ', '') |
regex_replace('[^A-Za-z0-9_]', '_')
  prefix: foreman_content_view_
  separator: ''
legacy_hostvars: true
plugin: theforeman.foreman.foreman
validate_certs: false
want_facts: true
want_hostcollections: false
want_params: true
```

15.6. OPENSTACK

```
expand_hostvars: true
fail_on_errors: true
inventory_hostname: uuid
plugin: openstack.cloud.openstack
```

15.7. RED HAT VIRTUALIZATION

```
compose:
  ansible_host: (devices.values() | list)[0][0] if devices else None
keyed_groups:
- key: cluster
  prefix: cluster
  separator: _
- key: status
  prefix: status
  separator: _
- key: tags
  prefix: tag
  separator: _
ovirt_hostname_preference:
- name
- fqdn
ovirt_insecure: false
plugin: ovirt.ovirt.ovirt
```

15.8. RED HAT ANSIBLE AUTOMATION PLATFORM

```
include_metadata: true
inventory_id: <inventory_id or url_quoted_named_url>
plugin: awx.awx.tower
validate_certs: <true or false>
```

CHAPTER 16. HOSTS

A host is a system managed by Ansible Automation Platform, which may include a physical, virtual, cloud-based server, or other device.

Typically a host is an operating system instance.

Hosts are grouped in inventories and are sometimes referred to as a “nodes”.

Ansible works against multiple managed nodes or “hosts” in your infrastructure at the same time, using a list or group of lists known as an inventory.

Once your inventory is defined, use patterns to select the hosts or groups you want Ansible to run against.

16.1. CREATING A HOST

To create a new host.

Procedure

1. From the navigation panel, select **Automation Execution → Infrastructure → Hosts**.
2. Click **Create host**.
3. On the **Create Host** page enter the following information:
   - **Name**: Enter a name for your host.
   - (Optional) **Description**: Enter a description for your host.
   - **Variables**: Enter the inventory file variables associated with your host.
4. Click **Create host** to save your changes.

16.2. VIEWING THE HOST DETAILS

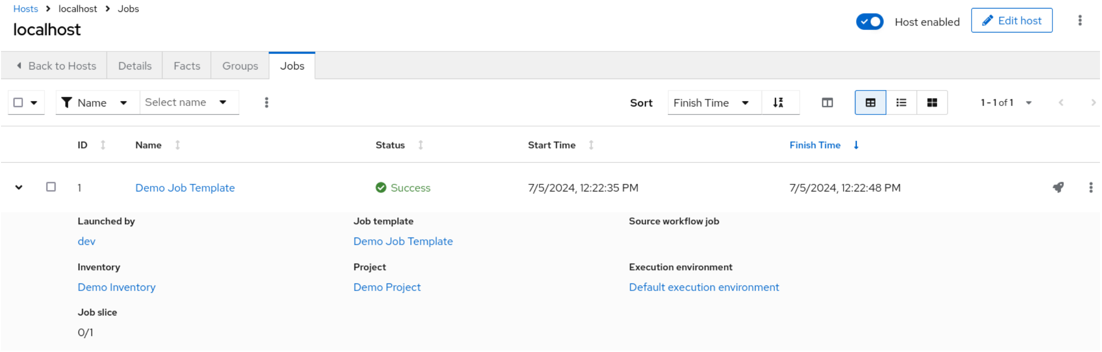
To view the Host details for a job run.

Procedure

1. From the navigation panel, select **Automation Execution → Infrastructure → Hosts**. The **Hosts** page displays the following information about the host or hosts affected by recent job runs.
2. Selecting a particular host displays the **Details** page for that host, with the following information:
   - The **Name** of the Host.
   - The **Inventory** associated with that host. Selecting this inventory displays details of the inventory.
   - When the Host was **Created** and who by. Selecting the creator displays details of the creator.

- When the Host was **Last modified**. Selecting the creator displays details of the creator.
- **Variables** associated with the Host. You can display the variables in YAML or JSON format.

3. Click **Edit host** to edit details of the host.
   - Select the **Facts** tab to display facts associated with the host.
   - Select the **Groups** tab to display the Groups associated with the host.
     - Click **Associate groups** to associate a group with the host.
   - Select the **Jobs** tab to display the Jobs which ran on the host.
     - Click the **>** icon to display details of the job.

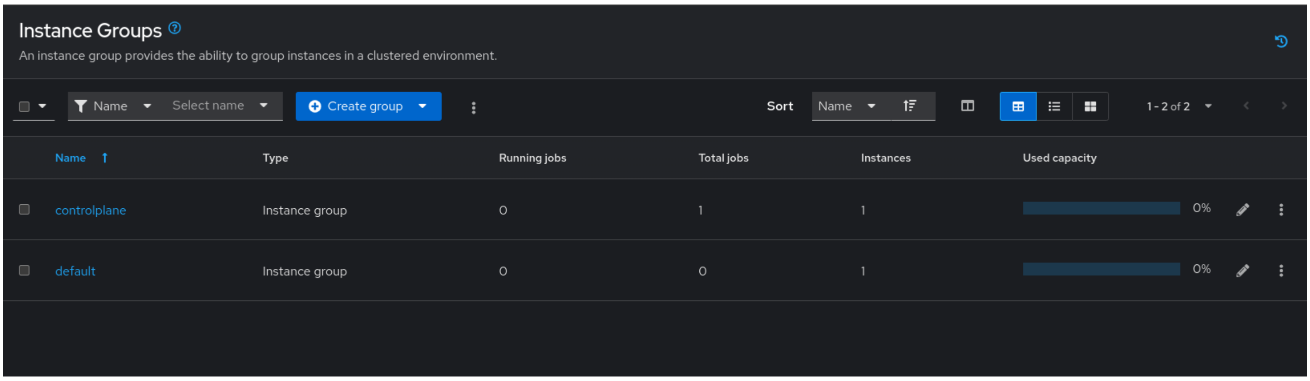


ID	Name	Status	Start Time	Finish Time
1	Demo Job Template	Success	7/5/2024, 12:22:35 PM	7/5/2024, 12:22:48 PM

Launched by: dev
Job template: Demo Job Template
Source workflow job
Inventory: Demo Inventory
Project: Demo Project
Execution environment: Default execution environment
Job slice: 0/1

CHAPTER 17. MANAGING INSTANCE GROUPS

An Instance Group enables you to group instances in a clustered environment. Policies dictate how instance groups behave and how jobs are executed. The following view displays the capacity levels based on policy algorithms:



Instance Groups

An instance group provides the ability to group instances in a clustered environment.

Name	Type	Running jobs	Total jobs	Instances	Used capacity
controlplane	Instance group	0	1	1	0%
default	Instance group	0	0	1	0%

Additional resources

- For more information about the policy or rules associated with instance groups, see the Instance Groups section of the *Configuring automation execution*.
- For more information about connecting your instance group to a container, see Container Groups.

17.1. CREATING AN INSTANCE GROUP

Use the following procedure to create a new instance group.

Procedure

1. From the navigation panel, select **Automation Execution → Infrastructure → Instance Groups**.
2. Click **Create group** and select **Create instance group** from the list.
3. Enter the appropriate details into the following fields:
   - **Name**: Names must be unique and must not be named "controller".
   - **Policy instance minimum**: Enter the minimum number of instances to automatically assign to this group when new instances come online.
   - **Policy instance percentage**: Use the slider to select a minimum percentage of instances to automatically assign to this group when new instances come online.

     NOTE

     Policy instance fields are not required to create a new instance group. If you do not specify values, then the **Policy instance minimum** and **Policy instance percentage** default to 0.

   - **Max concurrent jobs**: Specify the maximum number of forks that can be run for any given job.

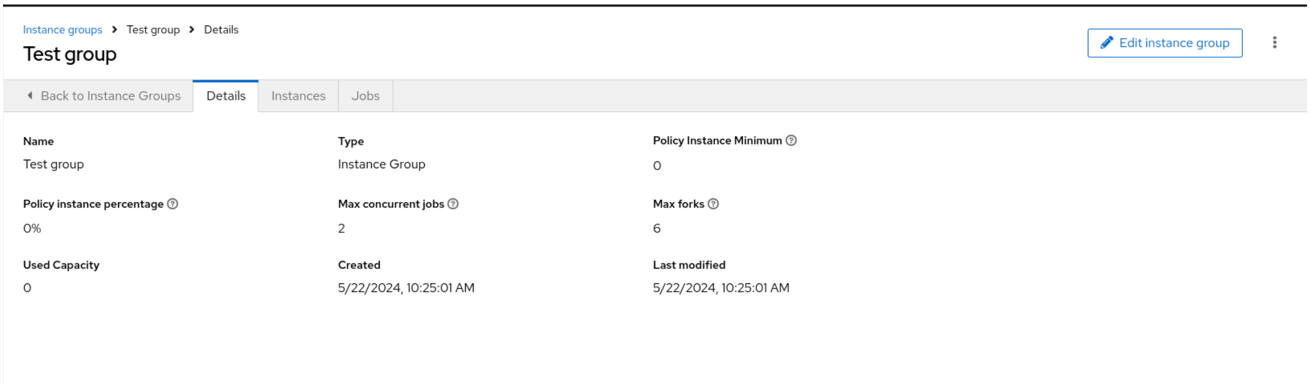
- **Max forks**: Specify the maximum number of concurrent jobs that can be run for any given job.

> NOTE
>
> The default value of 0 for **Max concurrent jobs** and **Max forks** denotes no limit. For more information, see Instance group capacity limits .

4. Click **Create instance group**, or, if you have edited an existing Instance Group click **Save instance group**

When you have successfully created the instance group the **Details** tab of the newly created instance group enables you to review and edit your instance group information.

You can also edit **Instances** and review **Jobs** associated with this instance group:



Instance groups › Test group › Details

Test group

Edit instance group

◂ Back to Instance Groups | Details | Instances | Jobs

Name	Type	Policy Instance Minimum
Test group	Instance Group	0
Policy instance percentage	**Max concurrent jobs**	**Max forks**
0%	2	6
Used Capacity	**Created**	**Last modified**
0	5/22/2024, 10:25:01 AM	5/22/2024, 10:25:01 AM

17.1.1. Associating instances to an instance group

Procedure

1. Select the **Instances** tab on the **Details** page of an Instance Group.
2. Click **Associate instance**.
3. Click the checkbox next to one or more available instances from the list to select the instances you want to associate with the instance group and click **Confirm**

17.1.2. Viewing jobs associated with an instance group

Procedure

1. Select the **Jobs** tab of the **Instance Group** window.
2. Click the arrow ❯ icon next to a job to expand the view and show details about each job. Each job displays the following details:
   - The job status
   - The ID and name
   - The type of job

- The time it started and completed
- Who started the job and applicable resources associated with it, such as the template, inventory, project, and execution environment

Additional resources

The instances are run in accordance with instance group policies. For more information, see Instance Group Policies.

CHAPTER 18. INSTANCE AND CONTAINER GROUPS

Automation controller enables you to execute jobs through Ansible playbooks run directly on a member of the cluster or in a namespace of an OpenShift cluster with the necessary service account provisioned. This is called a container group. You can execute jobs in a container group only as-needed per playbook. For more information, see Container groups.

For execution environments, see Execution environments.

18.1. INSTANCE GROUPS

Instances can be grouped into one or more instance groups. Instance groups can be assigned to one or more of the following listed resources:

- Organizations
- Inventories
- Job templates

When a job associated with one of the resources executes, it is 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. Instance groups associated with inventories are checked before those associated with organizations. Therefore, instance group assignments for the three resources form the hierarchy:

Job Template > Inventory > Organization

Consider the following when working with instance groups:

- You can define other groups and group instances in those groups. These groups must 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 appears as **controlplane** in the API instance groups. For more information and example scenarios, see Group policies for **automationcontroller**.
- You cannot modify the **controlplane** instance group, and attempting to do so results in a permission denied error for any user.
  Therefore, the **Disassociate** option is not available in the **Instances** tab of **controlplane**.
- A **default** API instance group is automatically created with all nodes capable of running jobs. This is like any other instance group but if a specific instance group is not associated with a specific resource, then the job execution always falls back to the default instance group. The default instance group always exists, and you cannot delete or rename it.
- Do not create a group named **instance_group_default**.
- Do not name any instance the same as a group name.

18.1.1. Group policies for automationcontroller

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. You can also create custom instance groups in the API after the install has finished.

The current behavior expects a member of an **instance_group_*** to be part of **automationcontroller** or **execution_nodes** group.

Example

```
[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
```

After you run installation program, the following error appears:

```
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, 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"}}}
```

After you upgrade from automation controller 4.0 or earlier, the legacy **instance_group_** member likely has the awx code installed. This places that node in the **automationcontroller** group.

18.1.2. Configure instance groups from the API

You can create instance groups by POSTing to **/api/v2/instance_groups** as a system administrator.

Once created, you can associate instances with an instance group using:

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

An instance that is added to an instance group automatically reconfigures itself to listen on the group's work queue. For more information, see the following section *Instance group policies*.

18.1.3. Instance group policies

You can configure automation 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.

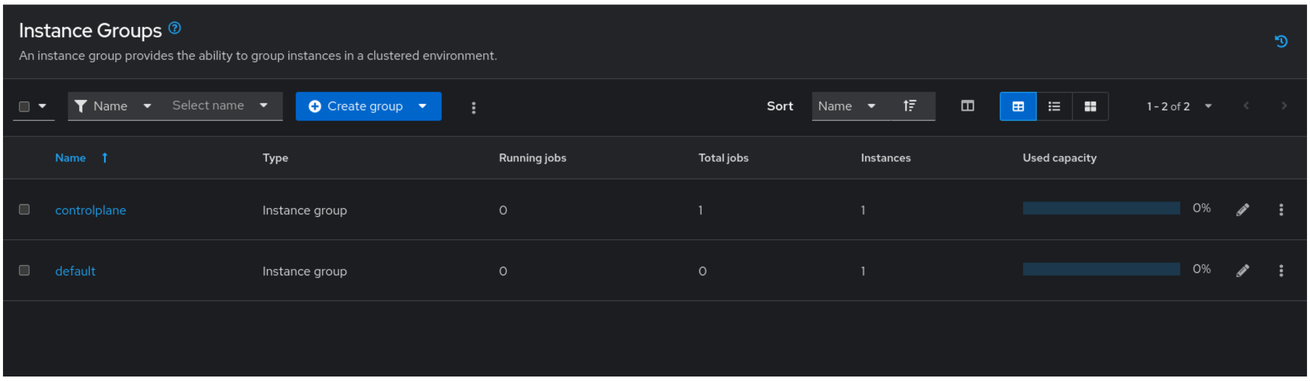
Instance group policies are controlled by the following three optional fields on an **Instance Group**:

- **policy_instance_percentage**: This is a number between 0 - 100. It guarantees that this percentage of active automation controller instances are 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 are 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 are 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 (UI) provides a summary of the capacity levels for each instance group according to instance group policies:



Instance Groups

An instance group provides the ability to group instances in a clustered environment.

	Name	Type	Running jobs	Total jobs	Instances	Used capacity
□	controlplane	Instance group	0	1	1	0%
□	default	Instance group	0	0	1	0%

Additional resources

For more information, see the Managing Instance Groups section.

18.1.4. Notable policy considerations

Take the following policy considerations into account:

- Both **policy_instance_percentage** and **policy_instance_minimum** set minimum allocations. The rule that results in more instances assigned to the group takes 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 are assigned to the instance group. If you reduce the number of total instances in the cluster to 2, then both of them are assigned to the instance group to satisfy **policy_instance_minimum**. This enables you to set a lower limit on the amount of available resources.

- Policies do not actively prevent instances from being associated with multiple instance groups, but this can 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 are distributed among them without any overlap.

18.1.5. Pinning instances manually to specific groups

If you have a special instance which needs to be only assigned to a specific instance group but do not want it to automatically join other groups by "percentage" or "minimum" policies:

Procedure

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 prevents the instance from being automatically added to other groups based on percentage and minimum policy. It only belongs to the groups you have 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
}
```

18.1.6. Job runtime behavior

When you run a job associated with an instance group, note the following behaviors:

- If you divide a cluster into separate instance groups, then the behavior is similar to the cluster as a whole. If you assign two instances to a group then either one is as likely to receive a job as any other in the same group.

- As automation controller instances are brought online, it effectively expands the work capacity of the system. If you place those instances 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 is reduced from all groups for which it is a member. De-provisioning an instance removes capacity from the cluster wherever that instance was assigned. For more information, see the Deprovisioning instance groups section for more detail.

NOTE

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

18.1.7. Control where a job runs

If you associate instance groups with a job template, inventory, or organization, a job run from that job template is not eligible for the default behavior. This means that if all of the instances inside of the instance groups associated with these three resources are out of capacity, the job remains 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 you associate instance groups with the job template, and all of these are at capacity, then the job is submitted to instance groups specified on the inventory, and then the organization. Jobs must execute in those groups in preferential order as resources are available.

You can still associate the global **default** group with a resource, such as any of the custom instance groups defined in the playbook. You can use this to specify a preferred instance group on the job template or inventory, but still enable the job to be submitted to any instance if those are out of capacity.

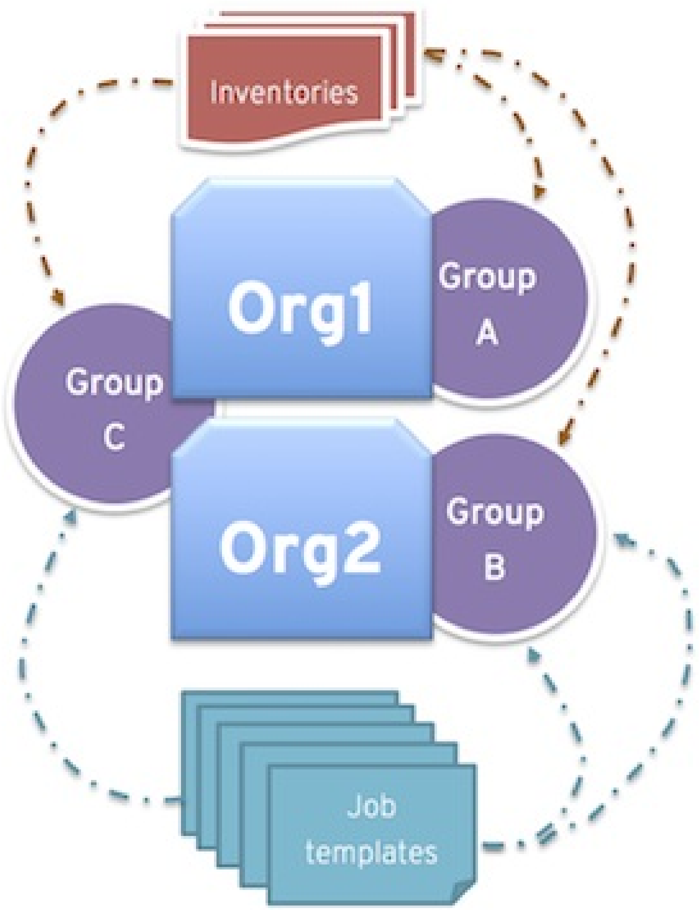
Examples

- If you associate **group_a** with a job template and also associate the **default** group with its inventory, you enable 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 choose another resource as the fallback. For example, not associating an instance group with a job template and having it fall back to the inventory or the organization's instance group.

This presents the following two examples:

1. Associating instance groups with an inventory (omitting assigning the job template to an instance group) ensures that any playbook run against a specific inventory runs only on the group associated with it. This is 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 enables 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.
   An administrator can assign multiple groups to each organization, similar to 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 assigns group C to both *Org1* and *Org2* as an overflow for any extra capacity that might be needed.

- The organization administrators are then free to assign inventory or job templates to whichever group they want, or let them inherit the default order from the organization.



Arranging resources this way offers you flexibility. You can also create instance groups with only one instance, enabling you to direct work towards a very specific Host in the automation controller cluster.

18.1.8. Instance group capacity limits

There is external business logic that can drive the need to limit the concurrency of jobs sent to an instance group, or the maximum number of forks to be consumed.

For traditional instances and instance groups, you might want to enable two organizations to run jobs on the same underlying instances, but limit each organization’s total number of concurrent jobs. This can be achieved by creating an instance group for each organization and assigning the value for **max_concurrent_jobs**.

For automation controller groups, automation controller is generally not aware of the resource limits of the OpenShift cluster. You can set limits on the number of pods on a namespace, or only resources available to schedule a certain number of pods at a time if no auto-scaling is in place. In this case, you can adjust the value for **max_concurrent_jobs**.

Another parameter available is **max_forks**. This provides additional flexibility for capping the capacity consumed on an instance group or container group. You can use this if jobs with a wide variety of inventory sizes and "forks" values are being run. You can limit an organization to run up to 10 jobs concurrently, but consume no more than 50 forks at a time:

```
max_concurrent_jobs: 10
max_forks: 50
```

If 10 jobs that use 5 forks each are run, an eleventh job waits until one of these finishes to run on that group (or be scheduled on a different group with capacity).

If 2 jobs are running with 20 forks each, then a third job with a **task_impact** of 11 or more waits until one of these finishes to run on that group (or be scheduled on a different group with capacity).

For container groups, using the **max_forks** value is useful given that all jobs are submitted using the same **pod_spec** with the same resource requests, irrespective of the "forks" value of the job. The default **pod_spec** sets requests and not limits, so the pods can "burst" above their requested value without being throttled or reaped. By setting the **max_forks value**, you can help prevent a scenario where too many jobs with large forks values get scheduled concurrently and cause the OpenShift nodes to be oversubscribed with multiple pods using more resources than their requested value.

To set the maximum values for the concurrent jobs and forks in an instance group, see Creating an instance group.

18.1.9. Deprovisioning instance groups

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

Procedure

1. Shut down the instance or stop the service with the following command:

   ```
   automation-controller-service stop
   ```

2. Run the following deprovision command from another instance to remove it from the controller cluster registry:

   ```
   awx-manage deprovision_instance --hostname=<name used in inventory file>
   ```

   Example

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

Deprovisioning instance groups in automation controller does not automatically deprovision or remove instance groups, even though re-provisioning often causes these to be unused. They can still show up in API endpoints and stats monitoring. You can remove these groups with the following command:

```
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 that the instance is not added back to a group. To be sure that an instance is not added back to a group, remove it through the API and also remove it in your inventory file. You can also stop defining instance groups in the inventory file. You can manage instance group topology through the automation controller UI. For more information about managing instance groups in the UI, see Managing Instance Groups.

NOTE

If you have isolated instance groups created in older versions of automation 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 isolated instances to execution nodes in the *Ansible Automation Platform Upgrade and Migration Guide*.

18.2. CONTAINER GROUPS

Ansible Automation Platform supports container groups, which enable you to execute jobs in automation controller regardless of whether automation 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. These 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, you might want to set container groups to be "always-on", which you can configure through the creation of an instance.

NOTE

Container groups upgraded from versions before automation controller 4.0 revert back to default and 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. For more information, see Execution environments.

18.2.1. Creating a container group

A **ContainerGroup** is a type of **InstanceGroup** that has an associated credential that enables you to connect to an OpenShift cluster.

Prerequisites

- A namespace that you can launch into. Every cluster has a "default" namespace, but you can use a specific namespace.

- A service account that has the roles that enable it to launch and manage pods in this namespace.

- If you are using execution environments in a private registry, and have a container registry credential associated with them in 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 must not have a container registry credential associated, or automation controller attempts to create the secret for you in the namespace.

- A token associated with that service account. An OpenShift or Kubernetes Bearer Token.

- A CA certificate associated with the cluster.

The following procedure explains how to create a service account in an OpenShift cluster or Kubernetes, to be used to run jobs in a container group through automation controller. After the service account is created, its credentials are provided to automation controller in the form of an OpenShift or Kubernetes API Bearer Token credential.

Procedure

1. To create a service account, download and use the sample service account, **containergroup sa** and modify it as needed to obtain the 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 certificate:

```
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, create an OpenShift or Kubernetes API Bearer Token credential to use with your container group. For more information, see Creating a credential.

Procedure

1. From the navigation panel, select **Automation Execution → Infrastructure → Instance Groups**.

2. Click **Create group** and select **Create container group**.

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

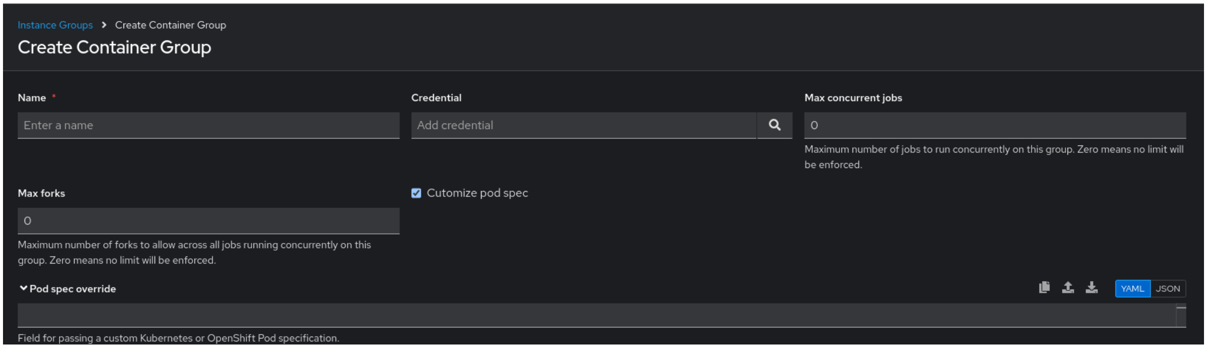
4. Click **Create container group**.

18.2.2. Customizing the pod specification

Ansible Automation Platform provides a simple default pod specification, however, you can provide a custom YAML or JSON document that overrides the default pod specification. This field uses any custom fields such as **ImagePullSecrets**, that can be "serialized" as valid pod JSON or YAML. A full list of options can be found in the Pods and Services section of the OpenShift documentation.

Procedure

1. From the navigation panel, select **Automation Execution → Infrastructure → Instance Groups**.
2. Click **Create group** and select **Create container group**.
3. Check the option for **Customize pod spec**.
4. Enter a custom Kubernetes or OpenShift Pod specification in the **Pod spec override** field.



Instance Groups > Create Container Group

Create Container Group

Name *: Enter a name

Credential: Add credential

Max concurrent jobs: 0

Maximum number of jobs to run concurrently on this group. Zero means no limit will be enforced.

Max forks: 0

Maximum number of forks to allow across all jobs running concurrently on this group. Zero means no limit will be enforced.

☑ Cutomize pod spec

Pod spec override

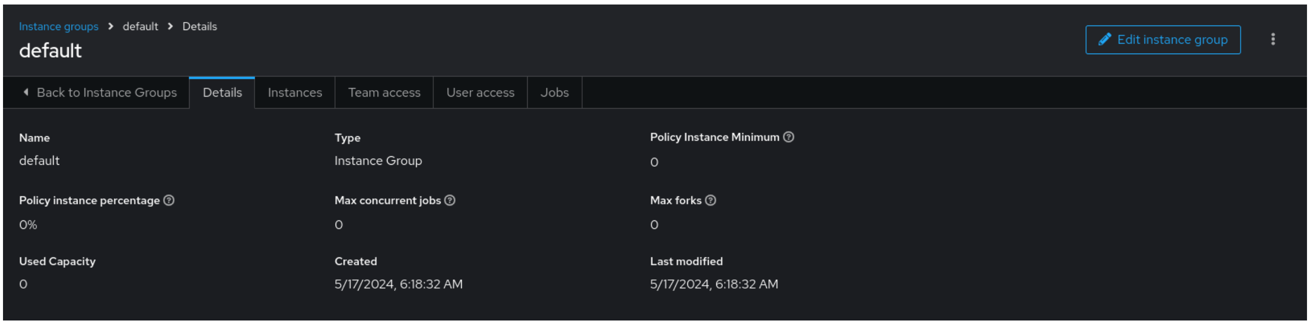
Field for passing a custom Kubernetes or OpenShift Pod specification.

5. Click **Create container group**.

> **NOTE**
>
> The image when a job launches is determined by which execution environment is associated with the job. If you associate a container registry credential with the execution environment, then automation controller attempts to make an **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 specification, and omit any credential from the execution environment used.
>
> For more information, see the Allowing Pods to Reference Images from Other Secured Registries section of the *Red Hat Container Registry Authentication* article.

Once you have created the container group successfully, the **Details** tab of the newly created container group remains, which enables you to review and edit your container group information. This is the same menu that is opened if you click the ✎ icon from the **Instance Groups** list view.

You can also edit **Instances** and review **Jobs** associated with this instance group.



Instance groups > default > Details

default

Name: default

Type: Instance Group

Policy Instance Minimum: 0

Policy instance percentage: 0%

Max concurrent jobs: 0

Max forks: 0

Used Capacity: 0

Created: 5/17/2024, 6:18:32 AM

Last modified: 5/17/2024, 6:18:32 AM

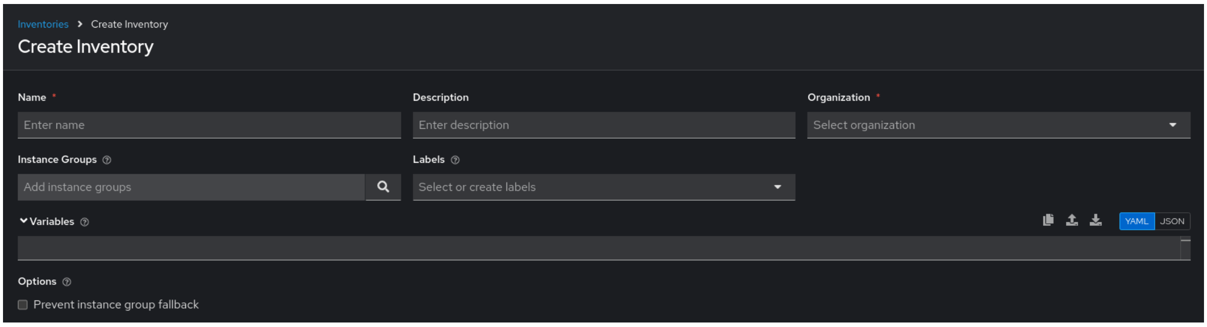
Container groups and instance groups are labeled accordingly.

18.2.3. Verifying container group functions

To verify the deployment and termination of your container:

Procedure

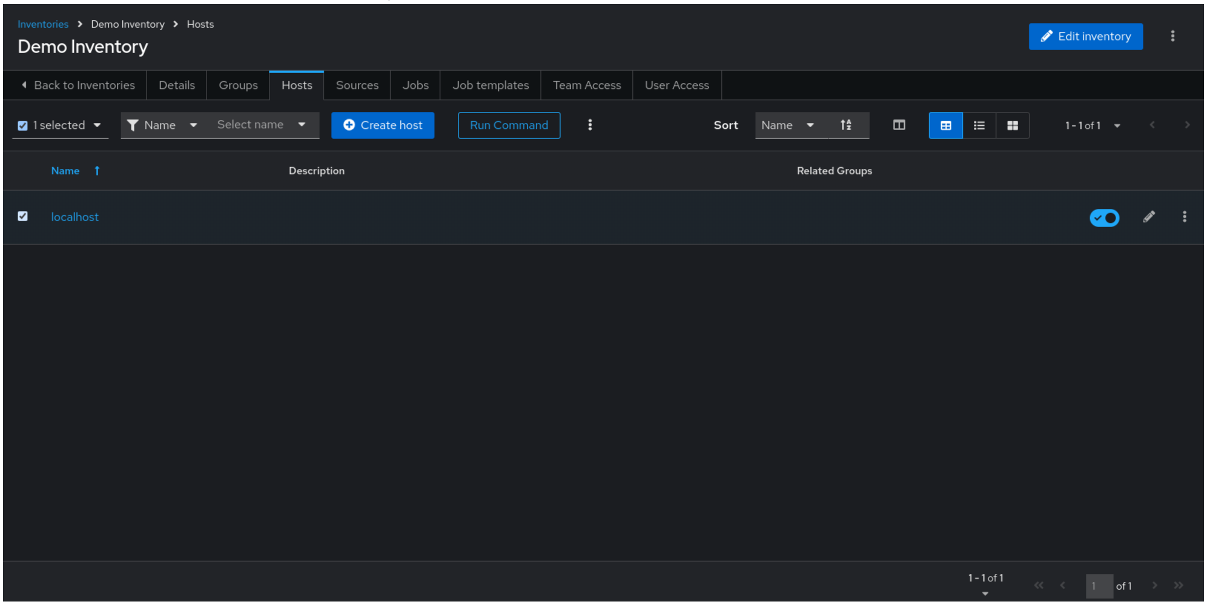
1. Create a mock inventory and associate the container group to it by populating the name of the container group in the **Instance groups** field. For more information, see Add a new inventory .

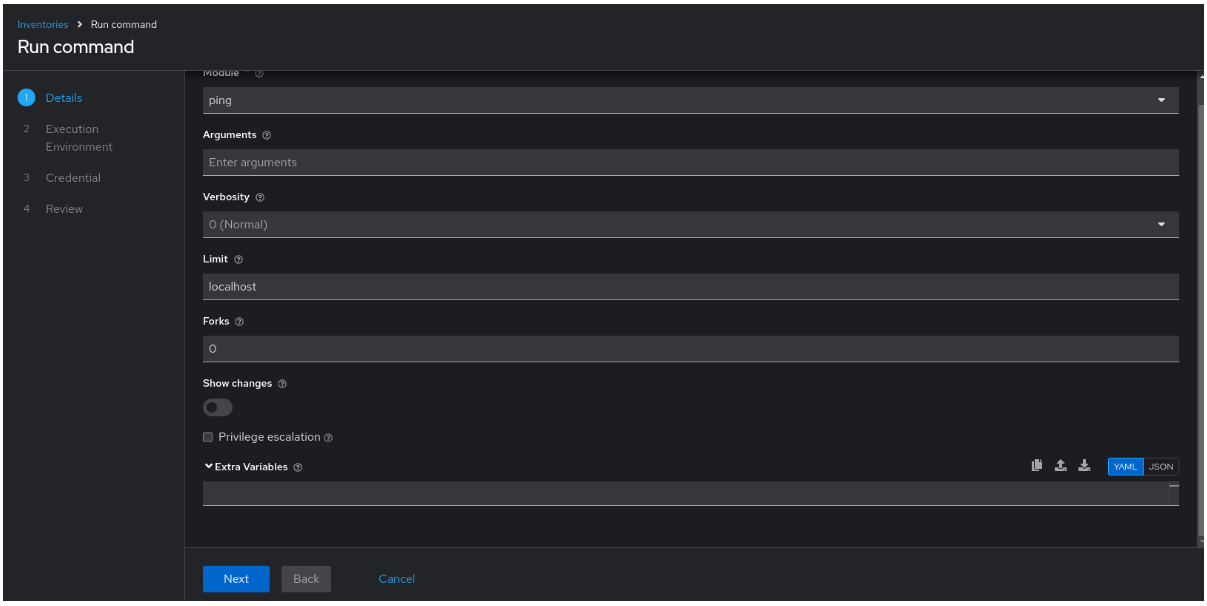


2. Create the **localhost** host in the inventory with the following variables:

> {'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 test:





Module: ping

Arguments: Enter arguments

Verbosity: 0 (Normal)

Limit: localhost

Forks: 0

Show changes

□ Privilege escalation

Extra Variables YAML JSON

You can see in the **Jobs** details view that the container was reached successfully by using one of the ad hoc jobs.

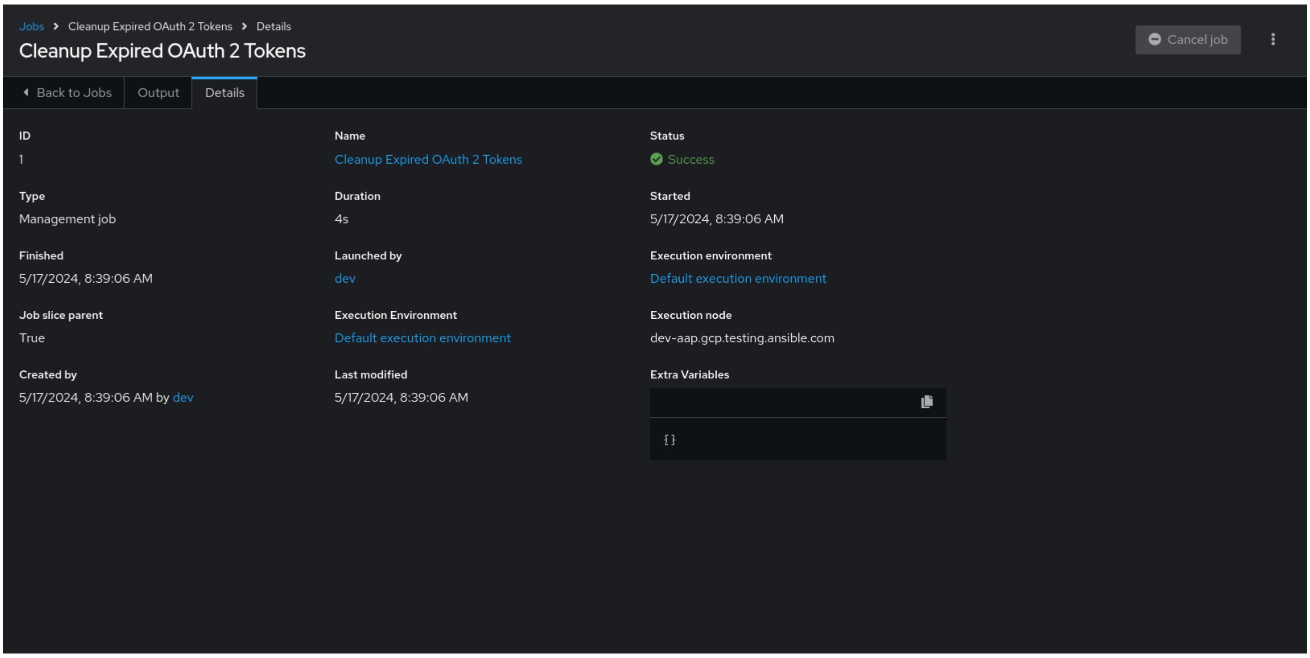
If you have an OpenShift UI, you can see pods appear and disappear as they deploy and end. You can also use the CLI to perform a **get pod** operation on your namespace to watch these same events occurring in real-time.

18.2.4. Viewing container group jobs

When you run a job associated with a container group, you can see the details of that job in the **Details** tab. You can also view its associated container group and the execution environment that spun up.

Procedure

1. From the navigation panel, select **Automation Execution → Jobs**.
2. Click a job for which you want to view a container group job.
3. Click the **Details** tab.



Cleanup Expired OAuth 2 Tokens

ID
1

Name
Cleanup Expired OAuth 2 Tokens

Status
Success

Type
Management job

Duration
4s

Started
5/17/2024, 8:39:06 AM

Finished
5/17/2024, 8:39:06 AM

Launched by
dev

Execution environment
Default execution environment

Job slice parent
True

Execution Environment
Default execution environment

Execution node
dev-aap.gcp.testing.ansible.com

Created by
5/17/2024, 8:39:06 AM by dev

Last modified
5/17/2024, 8:39:06 AM

Extra Variables

```
{}
```

18.2.5. Kubernetes API failure conditions

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

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

18.2.6. Container capacity limits

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

- To set limits on all pods within a given namespace, use the **LimitRange** object. For more information see the Quotas and Limit Ranges section of the OpenShift documentation.

- To set limits directly on the pod definition launched by automation controller, see Customizing the pod specification and the Compute Resources section of the OpenShift documentation.

NOTE

Container groups do not use the capacity algorithm that normal nodes use. You need to set the number of forks at the job template level. If you configure forks in automation controller, that setting is passed along to the container.

CHAPTER 19. MANAGING CAPACITY WITH INSTANCES

Scaling your automation mesh is available on OpenShift deployments of Red Hat Ansible Automation Platform and is possible through adding or removing nodes from your cluster dynamically, using the **Instances** resource of the UI, without running the installation script.

Instances serve as nodes in your mesh topology. Automation mesh enables you to extend the footprint of your automation. The location where you launch a job can be different from the location where the ansible-playbook runs.

To manage instances from the UI, you must have System Administrator or System Auditor permissions.

In general, the more processor cores (CPU) and memory (RAM) a node has, the more jobs that can be scheduled to run on that node at once.

For more information, see Automation controller capacity determination and job impact .

19.1. PREREQUISITES

The automation mesh is dependent on hop and execution nodes running on *Red Hat Enterprise Linux* (RHEL). Your Red Hat Ansible Automation Platform subscription grants you ten Red Hat Enterprise Linux licenses that can be used for running components of Ansible Automation Platform.

For more information about Red Hat Enterprise Linux subscriptions, see Registering the system and managing subscriptions in the Red Hat Enterprise Linux documentation.

The following steps prepare the RHEL instances for deployment of the automation mesh.

1. You require a Red Hat Enterprise Linux operating system. Each node in the mesh requires a static IP address, or a resolvable DNS hostname that Ansible Automation Platform can access.

2. Ensure that you have the minimum requirements for the RHEL virtual machine before proceeding. For more information, see the System requirements.

3. Deploy the RHEL instances within the remote networks where communication is required. For information about creating virtual machines, see Creating Virtual Machines in the *Red Hat Enterprise Linux* documentation. Remember to scale the capacity of your virtual machines sufficiently so that your proposed tasks can run on them.

    - RHEL ISOs can be obtained from access.redhat.com.

    - RHEL cloud images can be built using Image Builder from console.redhat.com.

19.2. PULLING THE SECRET

If you are using the default execution environment (provided with automation controller) to run on remote execution nodes, you must add a pull secret in the automation controller namespace that contains the credential for pulling the execution environment image.

To do this, create a pull secret on the automation controller namespace and configure the **ee_pull_credentials_secret** parameter in the Operator as follows:

Procedure

1. Create a secret:

```
oc create secret generic ee-pull-secret \
     --from-literal=username=<username> \
     --from-literal=password=<password> \
     --from-literal=url=registry.redhat.io


oc edit automationcontrollers <instance name>
```

2. Add **ee_pull_credentials_secret ee-pull-secret** to the specification:

```
spec.ee_pull_credentials_secret=ee-pull-secret
```

NOTE

To manage instances from the automation controller UI, you must have System administrator permissions.

19.3. SETTING UP VIRTUAL MACHINES FOR USE IN AN AUTOMATION MESH

Procedure

1. SSH into each of the RHEL instances and perform the following steps. Depending on your network access and controls, SSH proxies or other access models might be required.
   Use the following command:

   ```
   ssh [username]@[host_ip_address]
   ```

   For example, for an Ansible Automation Platform instance running on Amazon Web Services.

   ```
   ssh ec2-user@10.0.0.6
   ```

2. Create or copy an SSH key that can be used to connect from the hop node to the execution node in later steps. This can be a temporary key used just for the automation mesh configuration, or a long-lived key. The SSH user and key are used in later steps.

3. Enable your RHEL subscription with **baseos** and **appstream** repositories. Ansible Automation Platform RPM repositories are only available through subscription-manager, not the *Red Hat Update Infrastructure* (RHUI). If you attempt to use any other Linux footprint, including RHEL with RHUI, this causes errors.

   ```
   sudo subscription-manager register --auto-attach
   ```

   If Simple Content Access is enabled for your account, use:

   ```
   sudo subscription-manager register
   ```

   For more information about Simple Content Access, see Getting started with simple content access.

4. Enable Ansible Automation Platform subscriptions and the proper Red Hat Ansible Automation Platform channel:

For RHEL 8

```
# subscription-manager repos --enable ansible-automation-platform-2.5-for-rhel-8-x86_64-rpms
```

For RHEL 9

```
# subscription-manager repos --enable ansible-automation-platform-2.5-for-rhel-9-x86_64-rpms
```

For ARM

```
# subscription-manager repos --enable ansible-automation-platform-2.5-for-rhel-aarch64-rpms
```

5. Ensure the packages are up to date:

```
sudo dnf upgrade -y
```

6. Install the ansible-core packages on the machine where the downloaded bundle is to run:

```
sudo dnf install -y ansible-core
```

NOTE

Ansible core is required on the machine that runs the automation mesh configuration bundle playbooks. This document assumes that happens on the execution node. However, this step can be omitted if you run the playbook from a different machine. You cannot run directly from the control node, this is not currently supported, but future support expects that the control node has direct connectivity to the execution node.

19.4. MANAGING INSTANCES

To expand job capacity, create a standalone **execution node** that can be added to run alongside a deployment of automation controller. These execution nodes are not part of the automation controller Kubernetes cluster.

The control nodes run in the cluster connect and submit work to the execution nodes through Receptor.

These execution nodes are registered in automation controller as type **execution** instances, meaning they are only used to run jobs, not dispatch work or handle web requests as control nodes do.

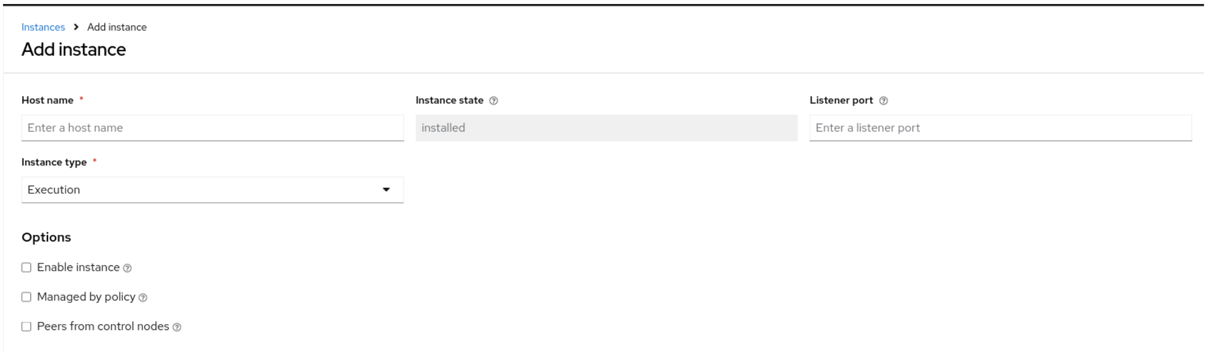
Hop nodes can be added to sit between the control plane of automation controller and standalone execution nodes. These hop nodes are not part of the Kubernetes cluster and are registered in automation controller as an instance of type **hop**, meaning they only handle inbound and outbound traffic for otherwise unreachable nodes in different or more strict networks.

The following procedure demonstrates how to set the node type for the hosts.

Procedure

1. From the navigation panel, select **Automation Execution → Infrastructure → Instances**.

2. On the **Instances** list page, click **Add instance**. The **Add Instance** window opens.



Instances > Add instance

Add instance

Host name *

Enter a host name

Instance state

installed

Listener port

Enter a listener port

Instance type *

Execution

Options

□ Enable instance

□ Managed by policy

□ Peers from control nodes

An instance requires the following attributes:

- **Host name**: (required) Enter a fully qualified domain name (public DNS) or IP address for your instance. This field is equivalent to **hostname** for installer-based deployments.

  NOTE

  If the instance uses private DNS that cannot be resolved from the control cluster, DNS lookup routing fails, and the generated SSL certificates is invalid. Use the IP address instead.

- Optional: **Description**: Enter a description for the instance.
- **Instance state**: This field is auto-populated, indicating that it is being installed, and cannot be modified.
- **Listener port**: This port is used for the receptor to listen on for incoming connections. You can set the port to one that is appropriate for your configuration. This field is equivalent to **listener_port** in the API. The default value is 27199, though you can set your own port value.
- **Instance type**: Only **execution** and **hop** nodes can be created. Operator based deployments do not support Control or Hybrid nodes.
  Options:
  - **Enable instance**: Check this box to make it available for jobs to run on an execution node.
  - Check the **Managed by policy** box to enable policy to dictate how the instance is assigned.
  - Check the **Peers from control nodes** box to enable control nodes to peer to this instance automatically. For nodes connected to automation controller, check the **Peers from Control** nodes box to create a direct communication link between that node and automation controller. For all other nodes:
    - If you are not adding a hop node, make sure **Peers from Control** is checked.
    - If you are adding a hop node, make sure **Peers from Control** is not checked.
    - For execution nodes that communicate with hop nodes, do not check this box.
  - To peer an execution node with a hop node, select the execution node, then select **Peers** tab.
    The Select Peers window is displayed.

    - Select the execution node to peer to the hop node.
    - Click **Associate peers**.

3. To view a graphical representation of your updated topology, see Topology view.

> NOTE
>
> Execute the following steps from any computer that has SSH access to the newly created instance.

4. Click the ⬇ icon next to **Download Bundle** to download the tar file that includes this new instance and the files necessary to install the created node into the automation mesh.
   The install bundle contains TLS certificates and keys, a certificate authority, and a proper Receptor configuration file.

```
receptor-ca.crt
work-public-key.pem
receptor.key
install_receptor.yml
inventory.yml
group_vars/all.yml
requirements.yml
```

5. Extract the downloaded **tar.gz** Install Bundle from the location where you downloaded it. To ensure that these files are in the correct location on the remote machine, the install bundle includes the **install_receptor.yml** playbook.

6. Before running the **ansible-playbook** command, edit the following fields in the **inventory.yml** file:

```
all:
  hosts:
    remote-execution:
      ansible_host: localhost # change to the mesh node host name
          ansible_user: <username> # user provided
          ansible_ssh_private_key_file: ~/.ssh/<id_rsa>
```

- Ensure **ansible_host** is set to the IP address or DNS of the node.
- Set **ansible_user** to the username running the installation.
- Set **ansible_ssh_private_key_file** to contain the filename of the private key used to connect to the instance.
- The content of the **inventory.yml** file serves as a template and contains variables for roles that are applied during the installation and configuration of a receptor node in a mesh topology. You can modify some of the other fields, or replace the file in its entirety for advanced scenarios. For more information, see Role Variables.

7. For a node that uses a private DNS, add the following line to **inventory.yml**:

```
ansible_ssh_common_args: <your ssh ProxyCommand setting>
```

This instructs the **install-receptor.yml** playbook to use the proxy command to connect through the local DNS node to the private node.

8. When the attributes are configured, click **Save**. The **Details** page of the created instance opens.

9. Save the file to continue.

10. The system that is going to run the install bundle to setup the remote node and run **ansible-playbook** requires the **ansible.receptor** collection to be installed:

    ```
    ansible-galaxy collection install ansible.receptor
    ```

    or

    ```
    ansible-galaxy install -r requirements.yml
    ```

    - Installing the receptor collection dependency from the **requirements.yml** file consistently retrieves the receptor version specified there. Additionally, it retrieves any other collection dependencies that might be needed in the future.

    - Install the receptor collection on all nodes where your playbook will run, otherwise an error occurs.

11. If **receptor_listener_port** is defined, the machine also requires an available open port on which to establish inbound TCP connections, for example, 27199. Run the following command to open port 27199 for receptor communication (Make sure you have port 27199 open in your firewall):

    ```
    sudo firewall-cmd --permanent --zone=public --add-port=27199/tcp
    ```

12. Run the following playbook on the machine where you want to update your automation mesh:

    ```
    ansible-playbook -i inventory.yml install_receptor.yml
    ```

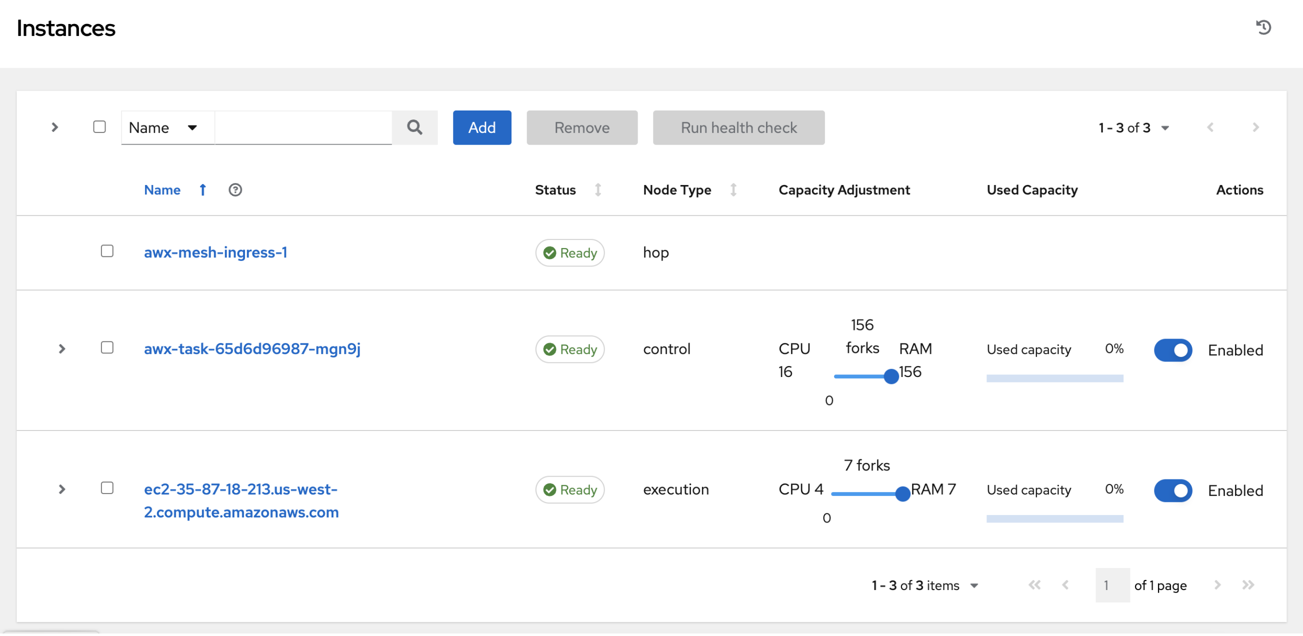
OpenSSL is required for this playbook. You can install it by running the following command:

```
openssl -v
```

If it returns then a version OpenSSL is installed. Otherwise you need to install OpenSSL with:

```
sudo dnf install -y openssl
```

After this playbook runs, your automation mesh is configured.



Name	Status	Node Type	Capacity Adjustment	Used Capacity	Actions
awx-mesh-ingress-1	Ready	hop			
awx-task-65d6d96987-mgn9j	Ready	control	CPU 16 156 forks RAM 156 0	Used capacity 0%	Enabled
ec2-35-87-18-213.us-west-2.compute.amazonaws.com	Ready	execution	CPU 4 7 forks RAM 7 0	Used capacity 0%	Enabled

To remove an instance from the mesh, see Removing instances.

19.5. REMOVING INSTANCES

From the **Instances** page, you can add, remove or run health checks on your nodes.

NOTE

You must follow the procedures for installing RHEL packages for any additional nodes you create. If you peer this additional node to an existing hop node, you must also install the Install Bundle on each node.

Use the check boxes next to an instance to select it to remove it, or run a health check against it.

NOTE

- If a node is removed using the UI, then the node is "removed" and does not show a status. If you delete the VM of the node before it is removed in the UI, it will show an error.
- You only need to reinstall the Install Bundle if the topology changes the communication pattern, that is, hop nodes change or you add nodes.

When a button is disabled, you do not have permission for that particular action. Contact your Administrator to grant you the required level of access.

If you are able to remove an instance, you receive a prompt for confirmation.

NOTE

You can still remove an instance even if it is active and jobs are running on it. Automation controller waits for jobs running on this node to complete before removing it.

CHAPTER 20. EXECUTION ENVIRONMENTS

Unlike legacy virtual environments, execution environments are container images that make it possible to incorporate system-level dependencies and collection-based content. Each execution environment enables you to have a customized image to run jobs and has only what is necessary when running the job.

20.1. BUILDING AN EXECUTION ENVIRONMENT

If your Ansible content depends on custom virtual environments instead of a default environment, you must take additional steps. You must install packages on each node, interact well with other software installed on the host system, and keep them in synchronization.

To simplify this process, you can build container images that serve as Ansible Control nodes. These container images are referred to as automation execution environments, which you can create with ansible-builder. Ansible-runner can then make use of those images.

20.1.1. Install ansible-builder

To build images, you must have Podman or Docker installed, along with the **ansible-builder** Python package.

The **--container-runtime** option must correspond to the Podman or Docker executable you intend to use.

When building an execution environment image, it must support the architecture that Ansible Automation Platform is deployed with.

For more information, see Quickstart for Ansible Builder, or Creating and consuming execution environments.

20.1.2. Content needed for an execution environment

Ansible-builder is used to create an execution environment.

An execution environment must contain:

- Ansible
- Ansible Runner
- Ansible Collections
- Python and system dependencies of:
  - modules or plugins in collections
  - content in ansible-base
  - custom user needs

Building a new execution environment involves a definition that specifies which content you want to include in your execution environment, such as collections, Python requirements, and system-level packages. The definition must be a .yml file

The content from the output generated from migrating to execution environments has some of the required data that can be piped to a file or pasted into this definition file.

Additional resources

For more information, see Migrate legacy venvs to execution environments . If you did not migrate from a virtual environment, you can create a definition file with the required data described in the Execution Environment Setup Reference.

Collection developers can declare requirements for their content by providing the appropriate metadata.

For more information, see Dependencies.

20.1.3. Example YAML file to build an image

The **ansible-builder build** command takes an execution environment definition as an input. It outputs the build context necessary for building an execution environment image, and then builds that image. The image can be re-built with the build context elsewhere, and produces the same result. By default, the builder searches for a file named **execution-environment.yml** in the current directory.

The following example **execution-environment.yml** file can be used as a starting point:

```
---
version: 3
dependencies:
  galaxy: requirements.yml
```

The content of **requirements.yml**:

```
---
collections:
  - name: awx.awx
```

To build an execution environment using the preceding files and run the following command:

```
ansible-builder build
...
STEP 7: COMMIT my-awx-ee
--> 09c930f5f6a
09c930f5f6ac329b7ddb321b144a029dbbfcc83bdfc77103968b7f6cdfc7bea2
Complete! The build context can be found at: context
```

In addition to producing a ready-to-use container image, the build context is preserved. This can be rebuilt at a different time or location with the tools of your choice, such as **docker build** or **podman build**.

Additional resources

For additional information about the **ansible-builder build** command, see Ansible's CLI Usage documentation.

20.1.4. Execution environment mount options

Rebuilding an execution environment is one way to add certificates, but inheriting certificates from the host provides a more convenient solution. For VM-based installations, automation controller automatically mounts the system truststore in the execution environment when jobs run.

You can customize execution environment mount options and mount paths in the **Paths to expose to isolated jobs** field of the **Job Settings** page, where Podman-style volume mount syntax is supported.

Additional resources

For more information, see the Podman documentation.

20.1.4.1. Troubleshooting execution environment mount options

In some cases where the **/etc/ssh/*** files were added to the execution environment image due to customization of an execution environment, an SSH error can occur. For example, exposing the **/etc/ssh/ssh_config.d:/etc/ssh/ssh_config.d:O** path enables the container to be mounted, but the ownership permissions are not mapped correctly.

Use the following procedure if you meet this error, or have upgraded from an older version of automation controller:

Procedure

1. Change the container ownership on the mounted volume to **root**.
2. From the navigation panel, select **Settings** → **Job**.
3. Click **Edit**.
4. Expose the path in the **Paths to expose to isolated jobs**field, using the current example:

   ```
   [
     "/ssh_config:/etc/ssh/ssh_config.d/:0"
   ]
   ```

   NOTE

   The **:O** option is only supported for directories. Be as detailed as possible, especially when specifying system paths. Mounting **/etc** or **/usr** directly has an impact that makes it difficult to troubleshoot.

   This informs Podman to run a command similar to the following example, where the configuration is mounted and the **ssh** command works as expected:

   ```
   podman run -v /ssh_config:/etc/ssh/ssh_config.d/:O ...
   ```

To expose isolated paths in OpenShift or Kubernetes containers as HostPath, use the following configuration:

```
[
  "/mnt2:/mnt2",
  "/mnt3",
  "/mnt4:/mnt4:0"
]
```

-

Set **Expose host paths for Container Groups**to **On** to enable it.

When the playbook runs, the resulting Pod specification is similar to the following example. Note the details of the **volumeMounts** and **volumes** sections.

```
apiVersion: v1
kind: Pod
spec:
  containers:
  - image: registry.redhat.io/ansible-automation-platform-22/ee-minimal-rhel8
    args:
      - ansible-runner
      - worker
      - --private-data-dir=/runner
    volumeMounts:
      - mountPath: /mnt2
        name: volume-0
        readOnly: true
      - mountPath: /mnt3
        name: volume-1
        readOnly: true
      - mountPath: /mnt4
        name: volume-2
        readOnly: true
  volumes:
    - hostPath:
        path: /mnt2
        type: ""
      name: volume-0
    - hostPath:
        path: /mnt3
        type: ""
      name: volume-1
    - hostPath:
        path: /mnt4
        type: ""
      name: volume-2
```

20.1.4.2. Mounting the directory in the execution node to the execution environment container

With Ansible Automation Platform 2.1.2, only **O** and **z** options were available. Since Ansible Automation Platform 2.2, further options such as **rw** are available. This is useful when using NFS storage.

Procedure

1. From the navigation panel, select **Settings → Job**.
2. Edit the **Paths to expose to isolated jobs**field:
   - Enter a list of paths that volumes are mounted from the execution node or the hybrid node to the container.
   - Enter one path per line.

- The supported format is **HOST-DIR[:CONTAINER-DIR[:OPTIONS]]**. The allowed paths are **z**, **O**, **ro**, and **rw**.

  Example

  ```
  [
    "/var/lib/awx/.ssh:/root/.ssh:O"
  ]
  ```

- For the **rw** option, configure the SELinux label correctly. For example, to mount the **/foo** directory, complete the following commands:

  ```
  sudo su
  ```

  ```
  mkdir /foo
  ```

  ```
  chmod 777 /foo
  ```

  ```
  semanage fcontext -a -t container_file_t "/foo(/.*)?"
  ```

  ```
  restorecon -vvFR /foo
  ```

The **awx** user must be permitted to read or write in this directory at least. Set the permissions as **777** at this time.

Additional resources

For more information about mount volumes, see the --volume option of the podman-run(1) section of the Podman documentation.

20.2. ADDING AN EXECUTION ENVIRONMENT TO A JOB TEMPLATE

Prerequisites

- An execution environment must have been created using ansible-builder as described in Build an execution environment. When an execution environment has been created, you can use it to run jobs. Use the automation controller UI to specify the execution environment to use in your job templates.
- Depending on whether an execution environment is made available for global use or tied to an organization, you must have the appropriate level of administrator privileges to use an execution environment in a job. Execution environments tied to an organization require Organization administrators to be able to run jobs with those execution environments.
- Before running a job or job template that uses an execution environment that has a credential assigned to it, ensure that the credential contains a username, host, and password.

Procedure

1. From the navigation panel, select **Automation Execution → Infrastructure → Execution Environments**.

2. Click **Create execution environment** to add an execution environment.

3. Enter the appropriate details into the following fields:

   - **Name** (required): Enter a name for the execution environment.

   - **Image** (required): Enter the image name. The image name requires its full location (repository), the registry, image name, and version tag in the example format of **quay.io/ansible/awx-ee:latestrepo/project/image-name:tag**.

   - Optional: **Pull**: Choose the type of pull when running jobs:

     - **Always pull container before running**: Pulls the latest image file for the container.

     - **Only pull the image if not present before running**: Only pulls the latest image if none is specified.

     - **Never pull container before running**: Never pull the latest version of the container image.

     NOTE

     If you do not set a type for pull, the value defaults to **Only pull the image if not present before running**.

   - Optional: **Description**:

   - Optional: **Organization**: Assign the organization to specifically use this execution environment. To make the execution environment available for use across multiple organizations, leave this field blank.

   - **Registry Credential**: If the image has a protected container registry, give the credential to access it.

4. Click **Create execution environment**.
   Your newly added execution environment is ready to be used in a job template.

5. To add an execution environment to a job template, specify it in the **Execution Environment** field of the job template.

When you have added an execution environment to a job template, those templates are listed in the **Templates** tab of the execution environment:

CHAPTER 21. EXECUTION ENVIRONMENT SETUP REFERENCE

This section contains reference information associated with the definition of an execution environment. You define the content of your execution environment in a YAML file. By default, this file is called **execution_environment.yml**. This file tells Ansible Builder how to create the build instruction file (Containerfile for Podman, Dockerfile for Docker) and build context for your container image.

NOTE

The definition schema for Ansible Builder 3.x is documented here. If you are running an older version of Ansible Builder, you need an older schema version. For more information, see older versions of this documentation. We recommend using version 3, which offers substantially more configurable options and functionality than previous versions.

21.1. EXECUTION ENVIRONMENT DEFINITION EXAMPLE

You must create a definition file to build an image for an execution environment. The file is in YAML format.

You must specify the version of Ansible Builder in the definition file. The default version is 1.

The following definition file is using Ansible Builder version 3:

```
version: 3
build_arg_defaults:
  ANSIBLE_GALAXY_CLI_COLLECTION_OPTS: '--pre'
dependencies:
  galaxy: requirements.yml
  python:
    - six
    - psutil
  system: bindep.txt
images:
  base_image:
    name: registry.redhat.io/ansible-automation-platform-24/ee-minimal-rhel8:latest
additional_build_files:
    - src: files/ansible.cfg
      dest: configs
additional_build_steps:
  prepend_galaxy:
    - ADD _build/configs/ansible.cfg /home/runner/.ansible.cfg
  prepend_final: |
    RUN whoami
    RUN cat /etc/os-release
  append_final:
    - RUN echo This is a post-install command!
    - RUN ls -la /etc
```

21.2. CONFIGURATION OPTIONS

Use the following configuration YAML keys in your definition file.

The Ansible Builder 3.x execution environment definition file accepts seven top-level sections:

- additional_build_files
- additional_build_steps
- build_arg_defaults
- dependencies
- images
    - image verification
- options
- version

21.2.1. additional_build_files

The build files specify what are to be added to the build context directory. These can then be referenced or copied by **additional_build_steps** during any build stage.

The format is a list of dictionary values, each with a **src** and **dest** key and value.

Each list item must be a dictionary containing the following required keys:

src	Specifies the source files to copy into the build context directory. This can be an absolute path, for example, **/home/user/.ansible.cfg**, or a path that is relative to the file. Relative paths can be a glob expression matching one or more files, for example, **files/*.cfg**. Note that an absolute path must not include a regular expression. If **src** is a directory, the entire contents of that directory are copied to **dest**.
dest	Specifies a subdirectory path underneath the **_build** subdirectory of the build context directory that contains the source files, for example, **files/configs**. This cannot be an absolute path or contain **..** within the path. This directory is created for you if it does not exist. NOTE When using an **ansible.cfg** file to pass a token and other settings for a private account to an automation hub server, listing the configuration file path here as a string enables it to be included as a build argument in the initial phase of the build.

21.2.2. additional_build_steps

The build steps specify custom build commands for any build phase. These commands are inserted directly into the build instruction file for the container runtime, for example, Containerfile or Dockerfile. The commands must conform to any rules required by the containerization tool.

You can add build steps before or after any stage of the image creation process. For example, if you need **git** to be installed before you install your dependencies, you can add a build step at the end of the base build stage.

The following are the valid keys. Each supports either a multi-line string, or a list of strings.

append_base	Commands to insert after building of the base image.
append_builder	Commands to insert after building of the builder image.
append_final	Commands to insert after building of the final image.
append_galaxy	Commands to insert after building of the galaxy image.
prepend_base	Commands to insert before building of the base image.
prepend_builder	Commands to insert before building of the builder image.
prepend_final	Commands to insert before building of the final image.
prepend_galaxy	Commands to insert before building of the galaxy image.

21.2.3. build_arg_defaults

This specifies the default values for build arguments as a dictionary.

This is an alternative to using the **--build-arg** CLI flag.

Ansible Builder uses the following build arguments:

ANSIBLE_GALAXY_CLI_COLLECTION_OPTS	Enables the user to pass the **-pre** flag and other flags to enable the installation of pre-release collections.
ANSIBLE_GALAXY_CLI_ROLE_OPTS	This enables the user to pass any flags, such as **--no-deps**, to the role installation.
PKGMGR_PRESERVE_CACHE	This controls how often the package manager cache is cleared during the image build process. If this value is not set, which is the default, the cache is cleared frequently. If the value is **always**, the cache is never cleared. Any other value forces the cache to be cleared only after the system dependencies are installed in the final build stage.

Ansible Builder hard-codes values given inside of **build_arg_defaults** into the build instruction file, so they persist if you run your container build manually.

If you specify the same variable in the definition and at the command line with the CLI **build-arg** flag, the CLI value overrides the value in the definition.

21.2.4. Dependencies

Specifies dependencies to install into the final image, including **ansible-core**, **ansible-runner**, Python packages, system packages, and collections. Ansible Builder automatically installs dependencies for any Ansible collections you install.

In general, you can use standard syntax to constrain package versions. Use the same syntax you would pass to **dnf**, **pip**, **ansible-galaxy**, or any other package management utility. You can also define your packages or collections in separate files and reference those files in the **dependencies** section of your definition file.

The following keys are valid:

<table>
<tr><td><b>ansible_core</b></td><td>The version of the <b>ansible-core</b> Python package to be installed.

This value is a dictionary with a single key, <b>package_pip</b>. The <b>package_pip</b> value is passed directly to pip for installation and can be in any format that pip supports. The following are some example values:
<pre>
ansible_core:
    package_pip: ansible-core
ansible_core:
    package_pip: ansible-core==2.14.3
ansible_core:
    package_pip: https://github.com/example_user/ansible/archive/refs/heads/ansible.tar.gz
</pre></td></tr>
<tr><td><b>ansible_runner</b></td><td>The version of the Ansible Runner Python package to be installed.

This value is a dictionary with a single key, <b>package_pip</b>. The <b>package_pip</b> value is passed directly to pip for installation and can be in any format that pip supports. The following are some example values:
<pre>
ansible_runner:
    package_pip: ansible-runner
ansible_runner:
    package_pip: ansible-runner==2.3.2
ansible_runner:
    package_pip: https://github.com/example_user/ansible-runner/archive/refs/heads/ansible-runner.tar.gz
</pre></td></tr>
<tr><td><b>galaxy</b></td><td>Collections to be installed from Ansible Galaxy.

This can be a filename, a dictionary, or a multi-line string representation of an Ansible Galaxy <b>requirements.yml</b> file. For more information about the requirements file format, see the Galaxy User Guide.</td></tr>
</table>

python	The Python installation requirements. This can be a filename, or a list of requirements. Ansible Builder combines all the Python requirements files from all collections into a single file using the **requirements-parser** library. This library supports complex syntax, including references to other files. If many collections require the same *package name*, Ansible Builder combines them into a single entry and combines the constraints. Ansible Builder excludes some packages in the combined file of Python dependencies even if a collection lists them as dependencies. These include test packages and packages that provide Ansible itself. The full list can is available under **EXCLUDE_REQUIREMENTS** in **src/ansible_builder/_target_scripts/introspect.py**. If you need to include one of these excluded package names, use the **--user-pip** option of the **introspect** command to list it in the user requirements file. Packages supplied this way are not processed against the list of excluded Python packages.
python_interpreter	A dictionary that defines the Python system package name to be installed by dnf (**package_system**) or a path to the Python interpreter to be used (**python_path)**.
system	The system packages to be installed, in bindep format. This can be a filename or a list of requirements. For more information about bindep, see the OpenDev documentation. For system packages, use the **bindep** format to specify cross-platform requirements, so they can be installed by whichever package management system the execution environment uses. Collections must specify necessary requirements for **[platform:rpm]**. Ansible Builder combines system package entries from multiple collections into a single file. Only requirements with no profiles (runtime requirements) are installed to the image. Entries from many collections which are duplicates of each other can be consolidated in the combined file.

The following example uses filenames that contain the various dependencies:

```
dependencies:
  python: requirements.txt
  system: bindep.txt
  galaxy: requirements.yml
  ansible_core:
      package_pip: ansible-core==2.14.2
  ansible_runner:
      package_pip: ansible-runner==2.3.1
```

```
python_interpreter:
   package_system: "python310"
   python_path: "/usr/bin/python3.10"
```

This example uses inline values:

```
dependencies:
  python:
    - pywinrm
  system:
    - iputils [platform:rpm]
  galaxy:
    collections:
      - name: community.windows
      - name: ansible.utils
        version: 2.10.1
  ansible_core:
      package_pip: ansible-core==2.14.2
  ansible_runner:
      package_pip: ansible-runner==2.3.1
  python_interpreter:
      package_system: "python310"
      python_path: "/usr/bin/python3.10"
```

NOTE

If any of these dependency files (**requirements.txt, bindep.txt, and requirements.yml**) are in the **build_ignore** of the collection, the build fails.

Collection maintainers can verify that ansible-builder recognizes the requirements they expect by using the **introspect** command:

```
ansible-builder introspect --sanitize ~/.ansible/collections/
```

The **--sanitize** option reviews all of the collection requirements and removes duplicates. It also removes any Python requirements that are normally excluded (see **python** dependencies).

Use the **-v3** option to **introspect** to see logging messages about requirements that are being excluded.

21.2.5. images

Specifies the base image to be used. At a minimum you must specify a source, image, and tag for the base image. The base image provides the operating system and can also provide some packages. Use the standard **host/namespace/container:tag** syntax to specify images. You can use Podman or Docker shortcut syntax instead, but the full definition is more reliable and portable.

Valid keys for this section are:

base_image	A dictionary defining the parent image for the execution environment. A **name** key must be supplied with the container image to use. Use the **signature_original_name** key if the image is mirrored within your repository, but signed with the original image's signature key.

21.2.6. Image verification

You can verify signed container images if you are using the **podman** container runtime.

Set the **container-policy** CLI option to control how this data is used in relation to a Podman **policy.json** file for container image signature validation.

- **ignore_all** policy: Generate a **policy.json** file in the build **context directory <context>** where no signature validation is performed.
- **system** policy: Signature validation is performed using pre-existing **policy.json** files in standard system locations. **ansible-builder** assumes no responsibility for the content within these files, and the user has complete control over the content.
- **signature_required** policy: **ansible-builder** uses the container image definitions to generate a **policy.json** file in the build **context directory <context>** that is used during the build to validate the images.

21.2.7. options

A dictionary of keywords or options that can affect the runtime functionality Ansible Builder.

Valid keys for this section are:

- **container_init**: A dictionary with keys that allow for customization of the container **ENTRYPOINT** and **CMD** directives (and related behaviors). Customizing these behaviors is an advanced task, and can result failures that are difficult to debug. Because the provided defaults control several intertwined behaviors, overriding any value skips all remaining defaults in this dictionary.
  Valid keys are:
  - **cmd**: Literal value for the **CMD** Containerfile directive. The default value is **["bash"]**.
  - **entrypoint**: Literal value for the **ENTRYPOINT** Containerfile directive. The default entrypoint behavior handles signal propagation to subprocesses, as well as attempting to ensure at runtime that the container user has a proper environment with a valid writeable home directory, represented in **/etc/passwd**, with the **HOME** environment variable set to match. The default entrypoint script can emit warnings to **stderr** in cases where it is unable to suitably adjust the user runtime environment. This behavior can be ignored or elevated to a fatal error; consult the source for the **entrypoint** target script for more details.
    The default value is **["/opt/builder/bin/entrypoint", "dumb-init"]**.
  - **package_pip**: Package to install with pip for entrypoint support. This package is installed in the final build image.
    The default value is **dumb-init==1.2.5**.
- **package_manager_path**: string with the path to the package manager (dnf or microdnf) to use. The default is **/usr/bin/dnf**. This value is used to install a Python interpreter, if specified in **dependencies**, and during the build phase by the **assemble** script.
- **skip_ansible_check**: This boolean value controls whether or not the check for an installation of Ansible and Ansible Runner is performed on the final image.
  Set this value to **True** to not perform this check.

  The default is **False**.

- **relax_passwd_permissions**: This boolean value controls whether the **root** group (GID 0) is explicitly granted write permission to **/etc/passwd** in the final container image. The default entrypoint script can attempt to update **/etc/passwd** under some container runtimes with dynamically created users to ensure a fully-functional POSIX user environment and home directory. Disabling this capability can cause failures of software features that require users to be listed in **/etc/passwd** with a valid and writeable home directory, for example, **async** in ansible-core, and the **~username** shell expansion.
  The default is **True**.

- **workdir**: Default current working directory for new processes started under the final container image. Some container runtimes also use this value as **HOME** for dynamically-created users in the **root** (GID 0) group. When this value is specified, if the directory does not already exist, it is created, set to **root** group ownership, and **rwx** group permissions are recursively applied to it.
  The default value is **/runner**.

- **user**: This sets the username or UID to use as the default user for the final container image.
  The default value is **1000**.

Example options:

```
options:
  container_init:
    package_pip: dumb-init>=1.2.5
    entrypoint: '["dumb-init"]'
    cmd: '["csh"]'
  package_manager_path: /usr/bin/microdnf
  relax_password_permissions: false
  skip_ansible_check: true
  workdir: /myworkdir
  user: bob
```

21.2.8. version

An integer value that sets the schema version of the execution environment definition file.

Defaults to **1**.

The value must be **3** if you are using Ansible Builder 3.x.

21.3. DEFAULT EXECUTION ENVIRONMENT FOR AWX

The example in **test/data/pytz** requires the **awx.awx** collection in the definition. The lookup plugin **awx.awx.tower_schedule_rrule** requires the PyPI **pytz** and another library to work. If the **test/data/pytz/execution-environment.yml** file is provided to the **ansible-builder build** command, it installs the collection inside the image, reads the **requirements.txt** file inside of the collection, and then installs **pytz** into the image.

The image produced can be used inside of an **ansible-runner** project by placing these variables inside the **env/settings** file, inside the private data directory.

```
---
container_image: image-name
process_isolation_executable: podman # or docker
process_isolation: true
```

The **awx.awx** collection is a subset of content included in the default AWX .

For further information, see the awx-ee repository.

CHAPTER 22. MANAGING USER CREDENTIALS

Credentials authenticate the automation controller user when launching jobs against machines, synchronizing with inventory sources, and importing project content from a version control system.

You can grant users and teams the ability to use these credentials, without exposing the credential to the user. If a user moves to a different team or leaves the organization, you do not have to re-key all of your systems just because that credential was available in automation controller.

NOTE

Automation controller encrypts passwords and key information in the database and never makes secret information visible through the API. For further information, see the *Configuring automation execution*.

22.1. HOW CREDENTIALS WORK

Automation controller uses SSH to connect to remote hosts. To pass the key from automation controller to SSH, the key must be decrypted before it can be written to a named pipe. Automation controller uses that pipe to send the key to SSH, so that the key is never written to disk. If passwords are used, automation controller handles them by responding directly to the password prompt and decrypting the password before writing it to the prompt.

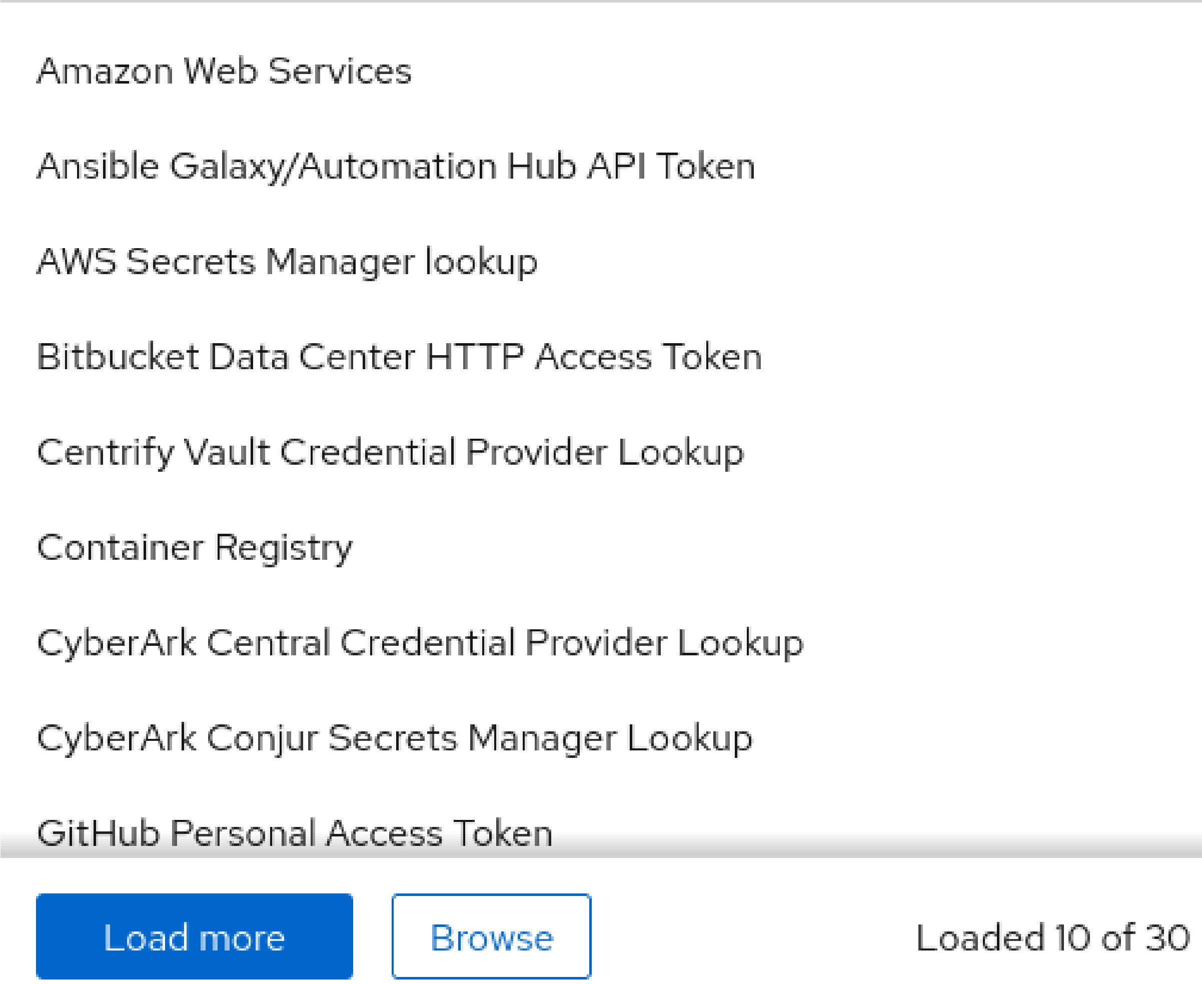
22.2. CREATING NEW CREDENTIALS

Credentials added to a team are made available to all members of the team. You can also add credentials to individual users.

As part of the initial setup, two credentials are available for your use: Demo Credential and Ansible Galaxy. Use the Ansible Galaxy credential as a template. You can copy this credential, but not edit it. Add more credentials as needed.

Procedure

1. From the navigation panel, select **Automation Execution → Infrastructure → Credentials**.
2. On the **Credentials** page, click **Create credential**.
3. Enter the following information:
   - **Name**: the name for your new credential.
   - (Optional) **Description**: a description for the new credential.
   - Optional **Organization**: The name of the organization with which the credential is associated. The default is **Default**.
   - **Credential type**: enter or select the credential type you want to create.
4. Enter the appropriate details depending on the type of credential selected, as described in Credential types.


Amazon Web Services
Ansible Galaxy/Automation Hub API Token
AWS Secrets Manager lookup
Bitbucket Data Center HTTP Access Token
Centrify Vault Credential Provider Lookup
Container Registry
CyberArk Central Credential Provider Lookup
CyberArk Conjur Secrets Manager Lookup
GitHub Personal Access Token
Load more
Browse
Loaded 10 of 30

5. Click **Create credential**.

22.3. ADDING NEW USERS AND JOB TEMPLATES TO EXISTING CREDENTIALS

Procedure

1. From the navigation panel, select **Automation Execution → Infrastructure → Credentials**.
2. Select the credential that you want to assign to additional users.
3. Click the **User Access** tab. You can see users and teams associated with this credential and their roles. If no users exist, add them from the **Users** menu. For more information, see Users.
4. Click **Add roles**.
5. Select the user(s) that you want to give access to the credential and click **Next**.
6. From the **Select roles to apply** page, select the roles you want to add to the User.
7. Click **Next**.
8. Review your selections and click **Finish** to add the roles or click **Back** to make changes. The **Add roles** window displays stating whether the action was successful.

   If the action is not successful, a warning displays.

9. Click **Close**.

10. The **User Access** page displays the summary information.

11. Select the **Job templates** tab to select a job template to which you want to assign this credential.

12. Chose a job template or select **Create job template** from the **Create template** list to assign the credential to additional job templates.
    For more information about creating new job templates, see the Job templates section.

22.4. CREDENTIAL TYPES

Automation controller supports the following credential types:

- Amazon Web Services
- Ansible Galaxy/Automation Hub API Token
- AWS Secrets Manager Lookup
- Bitbucket Data Center HTTP Access Token
- Centrify Vault Credential Provider Lookup
- Container Registry
- CyberArk Central Credential Provider Lookup
- CyberArk Conjur Secrets Manager Lookup
- GitHub Personal Access Token
- GitLab Personal Access Token
- Google Compute Engine
- GPG Public Key
- HashiCorp Vault Secret Lookup
- HashiCorp Vault Signed SSH
- Insights
- Machine
- Microsoft Azure Key Vault
- Microsoft Azure Resource Manager
- Network
- OpenShift or Kubernetes API Bearer Token
- OpenStack

- Red Hat Ansible Automation Platform
- Red Hat Satellite 6
- Red Hat Virtualization
- Source Control
- Terraform Backend Configuration
- Thycotic DevOps Secrets Vault
- Thycotic Secret Server
- Vault
- VMware vCenter

The credential types associated with AWS Secrets Manager, Centrify, CyberArk, HashiCorp Vault, Microsoft Azure Key Vault, and Thycotic are part of the credential plugins capability that enables an external system to lookup your secrets information.

For more information, see Secrets Management System.

22.4.1. Amazon Web Services credential type

Select this credential to enable synchronization of cloud inventory with Amazon Web Services.

Automation controller uses the following environment variables for AWS credentials:

```
AWS_ACCESS_KEY_ID
AWS_SECRET_ACCESS_KEY
AWS_SECURITY_TOKEN
```

These are fields prompted in the user interface.

Amazon Web Services credentials consist of the AWS **Access Key** and **Secret Key**.

Automation controller provides support for EC2 STS tokens, also known as Identity and Access Management (IAM) STS credentials. *Security Token Service* (STS) is a web service that enables you to request temporary, limited-privilege credentials for AWS IAM users.

NOTE

If the value of your tags in EC2 contain Booleans (**yes**/**no**/**true**/**false**), you must quote them.

WARNING

To use implicit IAM role credentials, do not attach AWS cloud credentials in automation controller when relying on IAM roles to access the AWS API.

Attaching your AWS cloud credential to your job template forces the use of your AWS credentials, not your IAM role credentials.

Additional resources

For more information about the IAM/EC2 STS Token, see Temporary security credentials in IAM .

22.4.1.1. Access Amazon EC2 credentials in an Ansible Playbook

You can get AWS credential parameters from a job runtime environment:

```
vars:
  aws:
    access_key: '{{ lookup("env", "AWS_ACCESS_KEY_ID") }}'
    secret_key: '{{ lookup("env", "AWS_SECRET_ACCESS_KEY") }}'
    security_token: '{{ lookup("env", "AWS_SECURITY_TOKEN") }}'
```

22.4.2. Ansible Galaxy/Automation Hub API token credential type

Select this credential to access Ansible Galaxy or use a collection published on an instance of private automation hub.

Entering the Galaxy server URL on this screen.

Populate the **Galaxy Server URL** field with the contents of the **Server URL** field at Red Hat Hybrid Cloud Console. Populate the **Auth Server URL** field with the contents of the **SSO URL** field at Red Hat Hybrid Cloud Console.

Additional resources

For more information, see Using Collections with automation hub .

22.4.3. AWS secrets manager lookup

This is considered part of the secret management capability. For more information, see AWS Secrets Manager Lookup

22.4.4. BitBucket data center HTTP access token

Bitbucket Data Center is a self-hosted Git repository for collaboration and management. Select this credential type to enable you to use HTTP access tokens in place of passwords for Git over HTTPS.

For further information, see HTTP access tokens in the Bitbucket Data Center documentation..

22.4.5. Centrify Vault Credential Provider Lookup credential type

This is considered part of the secret management capability. For more information, see Centrify Vault Credential Provider Lookup.

22.4.6. Container Registry credential type

Select this credential to enable automation controller to access a collection of container images. For more information, see What is a container registry? .

You must specify a name. The **Authentication URL** field is pre-populated with a default value. You can change the value by specifying the authentication endpoint for a different container registry.

22.4.7. CyberArk Central Credential Provider Lookup credential type

This is considered part of the secret management capability.

For more information, see CyberArk Central Credential Provider (CCP) Lookup .

22.4.8. CyberArk Conjur Secrets Manager Lookup credential type

This is considered part of the secret management capability.

For more information, see CyberArk Conjur Secrets Manager Lookup .

22.4.9. GitHub Personal Access Token credential type

Select this credential to enable you to access GitHub by using a *Personal Access Token* (PAT), which you can get through GitHub.

For more information, see Working with Webhooks.

GitHub PAT credentials require a value in the **Token** field, which is provided in your GitHub profile settings.

Use this credential to establish an API connection to GitHub for use in webhook listener jobs, to post status updates.

22.4.10. GitLab Personal Access Token credential type

Select this credential to enable you to access GitLab by using a *Personal Access Token* (PAT), which you can get through GitLab.

For more information, see Working with Webhooks.

GitLab PAT credentials require a value in the **Token** field, which is provided in your GitLab profile settings.

Use this credential to establish an API connection to GitLab for use in webhook listener jobs, to post status updates.

22.4.11. Google Compute Engine credential type

Select this credential to enable synchronization of a cloud inventory with Google Compute Engine (GCE).

Automation controller uses the following environment variables for GCE credentials:

```
GCE_EMAIL
GCE_PROJECT
GCE_CREDENTIALS_FILE_PATH
```

These are fields prompted in the user interface:

GCE credentials require the following information:

- **Service Account Email Address** The email address assigned to the Google Compute Engine **service account**.
- Optional: **Project**: Provide the GCE assigned identification or the unique project ID that you provided at project creation time.
- Optional: **Service Account JSON File**: Upload a GCE service account file. Click **Browse** to browse for the file that has the special account information that can be used by services and applications running on your GCE instance to interact with other Google Cloud Platform APIs. This grants permissions to the service account and virtual machine instances.
- **RSA Private Key**: The PEM file associated with the service account email.

22.4.11.1. Access Google Compute Engine credentials in an Ansible Playbook

You can get GCE credential parameters from a job runtime environment:

```
vars:
  gce:
    email: '{{ lookup("env", "GCE_EMAIL") }}'
    project: '{{ lookup("env", "GCE_PROJECT") }}'
    pem_file_path: '{{ lookup("env", "GCE_PEM_FILE_PATH") }}'
```

22.4.12. GPG Public Key credential type

Select this credential type to enable automation controller to verify the integrity of the project when synchronizing from source control.

For more information about how to generate a valid keypair, use the CLI tool to sign content, and how to add the public key to the controller, see Project Signing and Verification .

22.4.13. HashiCorp Vault Secret Lookup credential type

This is considered part of the secret management capability.

For more information, see HashiCorp Vault Secret Lookup .

22.4.14. HashiCorp Vault Signed SSH credential type

This is considered part of the secret management capability.

For more information, see HashiCorp Vault Signed SSH .

22.4.15. Insights credential type

Select this credential type to enable synchronization of cloud inventory with Red Hat Insights.

Insights credentials are the Insights **Username** and **Password**, which are the user's Red Hat Customer Portal Account username and password.

The **extra_vars** and **env** injectors for Insights are as follows:

```
ManagedCredentialType(
   namespace='insights',
....
....
....

injectors={
        'extra_vars': {
            "scm_username": "{{username}}",
            "scm_password": "{{password}}",
        },
        'env': {
            'INSIGHTS_USER': '{{username}}',
            'INSIGHTS_PASSWORD': '{{password}}',
        },
```

22.4.16. Machine credential type

Machine credentials enable automation controller to call Ansible on hosts under your management. You can specify the SSH username, optionally give a password, an SSH key, a key password, or have automation controller prompt the user for their password at deployment time. They define SSH and user-level privilege escalation access for playbooks, and are used when submitting jobs to run playbooks on a remote host.

The following network connections use **Machine** as the credential type: **httpapi**, **netconf**, and **network_cli**

Machine and SSH credentials do not use environment variables. They pass the username through the ansible **-u** flag, and interactively write the SSH password when the underlying SSH client prompts for it.

Machine credentials require the following inputs:

- **Username**: The username to use for SSH authentication.
- **Password**: The password to use for SSH authentication. This password is stored encrypted in the database, if entered. Alternatively, you can configure automation controller to ask the user for the password at launch time by selecting **Prompt on launch**. In these cases, a dialog opens when the job is launched, promoting the user to enter the password and password confirmation.
- **SSH Private Key**: Copy or drag-and-drop the SSH private key for the machine credential.
- **Private Key Passphrase**: If the SSH Private Key used is protected by a password, you can configure a Key Passphrase for the private key. This password is stored encrypted in the database, if entered. You can also configure automation controller to ask the user for the key passphrase at launch time by selecting **Prompt on launch**. In these cases, a dialog opens when the job is launched, prompting the user to enter the key passphrase and key passphrase confirmation.
- **Privilege Escalation Method**: Specifies the type of escalation privilege to assign to specific

users. This is the same as specifying the **--become-method=BECOME_METHOD** parameter, where **BECOME_METHOD** is any of the existing methods, or a custom method you have written. Begin entering the name of the method, and the appropriate name auto-populates.

- **empty selection**: If a task or play has **become** set to **yes** and is used with an empty selection, then it will default to **sudo**.
- **sudo**: Performs single commands with superuser (root user) privileges.
- **su**: Switches to the superuser (root user) account (or to other user accounts).
- **pbrun**: Requests that an application or command be run in a controlled account and provides for advanced root privilege delegation and keylogging.
- **pfexec**: Executes commands with predefined process attributes, such as specific user or group IDs.
- **dzdo**: An enhanced version of sudo that uses RBAC information in Centrify's Active Directory service. For more information, see Centrify's site on DZDO.
- **pmrun**: Requests that an application is run in a controlled account. See Privilege Manager for Unix 6.0.
- **runas**: Enables you to run as the current user.
- **enable**: Switches to elevated permissions on a network device.
- **doas**: Enables your remote/login user to run commands as another user through the *doas* ("Do as user") utility.
- **ksu**: Enables your remote/login user to run commands as another user through Kerberos access.
- **machinectl**: Enables you to manage containers through the **systemd** machine manager.
- **sesu**: Enables your remote/login user to run commands as another user through the CA Privileged Access Manager.

NOTE

Custom **become** plugins are available from Ansible 2.8+. For more information, see Understanding Privilege Escalation and the list of Become plugins

- **Privilege Escalation Username**: You see this field only if you selected an option for privilege escalation. Enter the username to use with escalation privileges on the remote system.
- **Privilege Escalation Password**: You see this field only if you selected an option for privilege escalation. Enter the password to use to authenticate the user through the selected privilege escalation type on the remote system. This password is stored encrypted in the database. You can also configure automation controller to ask the user for the password at launch time by selecting **Prompt on launch**. In these cases, a dialog opens when the job is launched, promoting the user to enter the password and password confirmation.

NOTE

You must use sudo password must in combination with SSH passwords or SSH Private Keys, because automation controller must first establish an authenticated SSH connection with the host before invoking **sudo** to change to the sudo user.

WARNING

Credentials that are used in scheduled jobs must not be configured as **Prompt on launch**.

22.4.16.1. Access machine credentials in an ansible playbook

You can get username and password from Ansible facts:

```
vars:
  machine:
    username: '{{ ansible_user }}'
    password: '{{ ansible_password }}'
```

22.4.17. Microsoft Azure Key Vault credential type

This is considered part of the secret management capability.

For more information, see Microsoft Azure Key Vault.

22.4.18. Microsoft Azure Resource Manager credential type

Select this credential type to enable synchronization of cloud inventory with Microsoft Azure Resource Manager.

Microsoft Azure Resource Manager credentials require the following inputs:

- **Subscription ID**: The Subscription UUID for the Microsoft Azure account.
- **Username**: The username to use to connect to the Microsoft Azure account.
- **Password**: The password to use to connect to the Microsoft Azure account.
- **Client ID**: The Client ID for the Microsoft Azure account.
- **Client Secret**: The Client Secret for the Microsoft Azure account.
- **Tenant ID**: The Tenant ID for the Microsoft Azure account.
- **Azure Cloud Environment** The variable associated with Azure cloud or Azure stack environments.

These fields are equal to the variables in the API.

To pass service principal credentials, define the following variables:

```
AZURE_CLIENT_ID
AZURE_SECRET
AZURE_SUBSCRIPTION_ID
AZURE_TENANT
AZURE_CLOUD_ENVIRONMENT
```

To pass an Active Directory username and password pair, define the following variables:

```
AZURE_AD_USER
AZURE_PASSWORD
AZURE_SUBSCRIPTION_ID
```

You can also pass credentials as parameters to a task within a playbook. The order of precedence is parameters, then environment variables, and finally a file found in your home directory.

To pass credentials as parameters to a task, use the following parameters for service principal credentials:

```
client_id
secret
subscription_id
tenant
azure_cloud_environment
```

Alternatively, pass the following parameters for Active Directory username/password:

```
ad_user
password
subscription_id
```

22.4.18.1. Access Microsoft Azure resource manager credentials in an ansible playbook

You can get Microsoft Azure credential parameters from a job runtime environment:

```
vars:
  azure:
    client_id: '{{ lookup("env", "AZURE_CLIENT_ID") }}'
    secret: '{{ lookup("env", "AZURE_SECRET") }}'
    tenant: '{{ lookup("env", "AZURE_TENANT") }}'
    subscription_id: '{{ lookup("env", "AZURE_SUBSCRIPTION_ID") }}'
```

22.4.19. Network credential type

> **NOTE**
>
> Select the Network credential type if you are using a *local* connection with *provider* to use Ansible networking modules to connect to and manage networking devices.
>
> When connecting to network devices, the credential type must match the connection type:

- For **local** connections using **provider**, credential type should be **Network**.

- For all other network connections (**httpapi**, **netconf**, and **network_cli**), the credential type should be **Machine**.

For more information about connection types available for network devices, see Multiple Communication Protocols.

Automation controller uses the following environment variables for Network credentials:

```
ANSIBLE_NET_USERNAME
ANSIBLE_NET_PASSWORD
```

Provide the following information for network credentials:

- **Username**: The username to use in conjunction with the network device.
- **Password**: The password to use in conjunction with the network device.
- **SSH Private Key**: Copy or drag-and-drop the actual SSH Private Key to be used to authenticate the user to the network through SSH.
- **Private Key Passphrase**: The passphrase for the private key to authenticate the user to the network through SSH.
- **Authorize**: Select this to control whether or not to enter privileged mode.
    - If **Authorize** is checked, enter a password in the **Authorize Password** field to access privileged mode.

For more information, see Porting Ansible Network Playbooks with New Connection Plugins .

22.4.20. Access network credentials in an ansible playbook

You can get the username and password parameters from a job runtime environment:

```
vars:
  network:
    username: '{{ lookup("env", "ANSIBLE_NET_USERNAME") }}'
    password: '{{ lookup("env", "ANSIBLE_NET_PASSWORD") }}'
```

22.4.21. OpenShift or Kubernetes API Bearer Token credential type

Select this credential type to create instance groups that point to a Kubernetes or OpenShift container.

For more information, see Instance and container groups.

Provide the following information for container credentials:

- **OpenShift or Kubernetes API Endpoint** (required): The endpoint used to connect to an OpenShift or Kubernetes container.
- **API authentication bearer token** (required): The token used to authenticate the connection.
- Optional: **Verify SSL**: You can check this option to verify the server's SSL/TLS certificate is valid and trusted. Environments that use internal or private *Certificate Authority* (CA) must leave this option unchecked to disable verification.

- **Certificate Authority data**: Include the **BEGIN CERTIFICATE** and **END CERTIFICATE** lines when pasting the certificate, if provided.

A container group is a type of instance group that has an associated credential that enables connection to an OpenShift cluster. To set up a container group, you must have the following items:

- A namespace you can start into. Although every cluster has a default namespace, you can use a specific namespace.
- A service account that has the roles that enable it to start and manage pods in this namespace.
- If you use execution environments in a private registry, and have a container registry credential associated with them in automation controller, the service account also requires 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 container group. In this case, the execution environment must not have a Container Registry credential associated, or automation controller attempts 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

22.4.21.1. Creating a service account in an Openshift cluster

Creating a service account in an Openshift or Kubernetes cluster to be used to run jobs in a container group through automation controller. After you create the service account, its credentials are provided to automation controller in the form of an Openshift or Kubernetes API bearer token credential.

After you create a service account, use the information in the new service account to configure automation controller.

Procedure

1. To create a service account, download and use the sample service account and change it as required to obtain the previous credentials.
2. Apply the configuration from the sample service account:

   ```
   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.

22.4.22. OpenStack credential type

Select this credential type to enable synchronization of cloud inventory with OpenStack.

Enter the following information for OpenStack credentials:

- **Username**: The username to use to connect to OpenStack.
- **Password (API Key)**: The password or API key to use to connect to OpenStack.
- **Host (Authentication URL)**: The host to be used for authentication.
- **Project (Tenant Name)**: The Tenant name or Tenant ID used for OpenStack. This value is usually the same as the username.
- Optional: **Project (Domain Name)**: Give the project name associated with your domain.
- Optional: **Domain Name**: Give the FQDN to be used to connect to OpenStack.
- Optional: **Region Name**: Give the region name. For some cloud providers, like OVH, the region must be specified.

If you are interested in using OpenStack Cloud Credentials, see Use Cloud Credentials with a cloud inventory, which includes a sample playbook.

22.4.23. Red Hat Ansible Automation Platform credential type

Select this credential to access another automation controller instance.

Ansible Automation Platform credentials require the following inputs:

- **Red Hat Ansible Automation Platform**: The base URL or IP address of the other instance to connect to.
- **Username**: The username to use to connect to it.
- **Password**: The password to use to connect to it.
- **Oauth Token**: If username and password are not used, provide an OAuth token to use to authenticate.

The **env** injectors for Ansible Automation Platform are as follows:

```
ManagedCredentialType(
    namespace='controller',

....
....
....

injectors={
        'env': {
            'TOWER_HOST': '{{host}}',
            'TOWER_USERNAME': '{{username}}',
            'TOWER_PASSWORD': '{{password}}',
            'TOWER_VERIFY_SSL': '{{verify_ssl}}',
            'TOWER_OAUTH_TOKEN': '{{oauth_token}}',
            'CONTROLLER_HOST': '{{host}}',
            'CONTROLLER_USERNAME': '{{username}}',
            'CONTROLLER_PASSWORD': '{{password}}',
            'CONTROLLER_VERIFY_SSL': '{{verify_ssl}}',
            'CONTROLLER_OAUTH_TOKEN': '{{oauth_token}}',
        }
```

22.4.23.1. Access automation controller credentials in an Ansible Playbook

You can get the host, username, and password parameters from a job runtime environment:

```
vars:
  controller:
    host: '{{ lookup("env", "CONTROLLER_HOST") }}'
    username: '{{ lookup("env", "CONTROLLER_USERNAME") }}'
    password: '{{ lookup("env", "CONTROLLER_PASSWORD") }}'
```

22.4.24. Red Hat Satellite 6 credential type

Select this credential type to enable synchronization of cloud inventory with Red Hat Satellite 6.

Automation controller writes a Satellite configuration file based on fields prompted in the user interface. The absolute path to the file is set in the following environment variable:

```
FOREMAN_INI_PATH
```

Satellite credentials have the following required inputs:

- **Satellite 6 URL**: The Satellite 6 URL or IP address to connect to.
- **Username**: The username to use to connect to Satellite 6.
- **Password**: The password to use to connect to Satellite 6.

22.4.25. Red Hat Virtualization credential type

Select this credential to enable automation controller to access Ansible’s **oVirt4.py** dynamic inventory plugin, which is managed by *Red Hat Virtualization*.

Automation controller uses the following environment variables for Red Hat Virtualization credentials. These are fields in the user interface:

```
OVIRT_URL
OVIRT_USERNAME
OVIRT_PASSWORD
```

Provide the following information for Red Hat Virtualization credentials:

- **Host (Authentication URL)**: The host URL or IP address to connect to. To sync with the inventory, the credential URL needs to include the **ovirt-engine/api** path.

- **Username**: The username to use to connect to oVirt4. This must include the domain profile to succeed, for example **username@ovirt.host.com**.

- **Password**: The password to use to connect to it.

- Optional: **CA File**: Provide an absolute path to the oVirt certificate file (it might end in **.pem**, **.cer** and **.crt** extensions, but preferably **.pem** for consistency)

22.4.25.1. Access virtualization credentials in an Ansible Playbook

You can get the Red Hat Virtualization credential parameter from a job runtime environment:

```
vars:
  ovirt:
    ovirt_url: '{{ lookup("env", "OVIRT_URL") }}'
    ovirt_username: '{{ lookup("env", "OVIRT_USERNAME") }}'
    ovirt_password: '{{ lookup("env", "OVIRT_PASSWORD") }}'
```

The **file** and **env** injectors for Red Hat Virtualization are as follows:

```
ManagedCredentialType(
    namespace='rhv',

....
....
....

injectors={
        # The duplication here is intentional; the ovirt4 inventory plugin
        # writes a .ini file for authentication, while the ansible modules for
        # ovirt4 use a separate authentication process that support
        # environment variables; by injecting both, we support both
        'file': {
            'template': '\n'.join(
                [
                    '[ovirt]',
                    'ovirt_url={{host}}',
                    'ovirt_username={{username}}',
                    'ovirt_password={{password}}',
                    '{% if ca_file %}ovirt_ca_file={{ca_file}}{% endif %}',
                ]
            )
        },
        'env': {'OVIRT_INI_PATH': '{{tower.filename}}', 'OVIRT_URL': '{{host}}', 'OVIRT_USERNAME':
'{{username}}', 'OVIRT_PASSWORD': '{{password}}'},
    },
)
```

22.4.26. Source Control credential type

Source Control credentials are used with projects to clone and update local source code repositories from a remote revision control system such as Git or Subversion.

Source Control credentials require the following inputs:

- **Username**: The username to use in conjunction with the source control system.
- **Password**: The password to use in conjunction with the source control system.
- **SCM Private Key**: Copy or drag-and-drop the actual SSH Private Key to be used to authenticate the user to the source control system through SSH.
- **Private Key Passphrase**: If the SSH Private Key used is protected by a passphrase, you can configure a Key Passphrase for the private key.

NOTE

You cannot configure Source Control credentials as **Prompt on launch**.

If you are using a GitHub account for a Source Control credential and you have *Two Factor Authentication* (2FA) enabled on your account, you must use your Personal Access Token in the password field rather than your account password.

22.4.27. Terraform backend configuration

Terraform is a HashiCorp tool used to automate various infrastructure tasks. Select this credential type to enable synchronization with the Terraform inventory source.

The Terraform credential requires the **Backend configuration** attribute which must contain the data from a Terraform backend block. You can paste, drag a file, browse to upload a file, or click the 🔑 icon to populate the field from an external Secret Management System.

Terraform backend configuration requires the following inputs:

- **Name**
- Credential type: Select **Terraform backend configuration**.
- Optional: **Organization**
- Optional: **Description**
- **Backend configuration**: Drag a file here or browse to upload.
  Example configuration for an S3 backend:

  ```
  bucket = "my-terraform-state-bucket"
  key = "path/to/terraform-state-file"
  region = "us-east-1"
  access_key = "my-aws-access-key"
  secret_key = "my-aws-secret-access-key"
  ```

- Optional: **Google Cloud Platform account credentials**

22.4.28. Thycotic DevOps Secrets Vault credential type

This is considered part of the secret management capability.

For more information, see Thycotic DevOps Secrets Vault.

22.4.29. Thycotic secret server credential type

This is considered part of the secret management capability.

For more information, see Thycotic Secret Server.

22.4.30. Ansible Vault credential type

Select this credential type to enable synchronization of inventory with Ansible Vault.

Vault credentials require the **Vault Password** and an optional **Vault Identifier** if applying multi-Vault credentialing.

You can configure automation controller to ask the user for the password at launch time by selecting **Prompt on launch**.

When you select **Prompt on launch**, a dialog opens when the job is launched, prompting the user to enter the password.

> ⚠ **WARNING**
>
> Credentials that are used in scheduled jobs must not be configured as **Prompt on launch**.

For more information about Ansible Vault, see Protecting sensitive data with Ansible vault .

22.4.31. VMware vCenter credential type

Select this credential type to enable synchronization of inventory with VMware vCenter.

Automation controller uses the following environment variables for VMware vCenter credentials:

```
VMWARE_HOST
VMWARE_USER
VMWARE_PASSWORD
VMWARE_VALIDATE_CERTS
```

These are fields prompted in the user interface.

VMware credentials require the following inputs:

- **vCenter Host**: The vCenter hostname or IP address to connect to.
- **Username**: The username to use to connect to vCenter.
- **Password**: The password to use to connect to vCenter.

NOTE

If the VMware guest tools are not running on the instance, VMware inventory synchronization does not return an IP address for that instance.

22.4.31.1. Access VMware vCenter credentials in an ansible playbook

You can get VMware vCenter credential parameters from a job runtime environment:

```
vars:
  vmware:
    host: '{{ lookup("env", "VMWARE_HOST") }}'
    username: '{{ lookup("env", "VMWARE_USER") }}'
    password: '{{ lookup("env", "VMWARE_PASSWORD") }}'
```

22.5. USE AUTOMATION CONTROLLER CREDENTIALS IN A PLAYBOOK

The following playbook is an example of how to use automation controller credentials in your playbook.

```
- hosts: all

  vars:
    machine:
      username: '{{ ansible_user }}'
      password: '{{ ansible_password }}'
    controller:
      host: '{{ lookup("env", "CONTROLLER_HOST") }}'
      username: '{{ lookup("env", "CONTROLLER_USERNAME") }}'
      password: '{{ lookup("env", "CONTROLLER_PASSWORD") }}'
    network:
      username: '{{ lookup("env", "ANSIBLE_NET_USERNAME") }}'
      password: '{{ lookup("env", "ANSIBLE_NET_PASSWORD") }}'
    aws:
      access_key: '{{ lookup("env", "AWS_ACCESS_KEY_ID") }}'
      secret_key: '{{ lookup("env", "AWS_SECRET_ACCESS_KEY") }}'
      security_token: '{{ lookup("env", "AWS_SECURITY_TOKEN") }}'
    vmware:
      host: '{{ lookup("env", "VMWARE_HOST") }}'
      username: '{{ lookup("env", "VMWARE_USER") }}'
      password: '{{ lookup("env", "VMWARE_PASSWORD") }}'
    gce:
      email: '{{ lookup("env", "GCE_EMAIL") }}'
      project: '{{ lookup("env", "GCE_PROJECT") }}'
    azure:
      client_id: '{{ lookup("env", "AZURE_CLIENT_ID") }}'
      secret: '{{ lookup("env", "AZURE_SECRET") }}'
      tenant: '{{ lookup("env", "AZURE_TENANT") }}'
      subscription_id: '{{ lookup("env", "AZURE_SUBSCRIPTION_ID") }}'

  tasks:
    - debug:
        var: machine
    - debug:
        var: controller

    - debug:
        var: network

    - debug:
        var: aws

    - debug:
        var: vmware

    - debug:
        var: gce

    - shell: 'cat {{ gce.pem_file_path }}'
      delegate_to: localhost

    - debug:
        var: azure
```

Use 'delegate_to' and any lookup variable

```
- command: somecommand
  environment:
    USERNAME: '{{ lookup("env", "USERNAME") }}'
    PASSWORD: '{{ lookup("env", "PASSWORD") }}'
  delegate_to: somehost
```

CHAPTER 23. CUSTOM CREDENTIAL TYPES

As a system administrator, you can define a custom credential type in a standard format by using a YAML or JSON-like definition. You can define a custom credential type that works in ways similar to existing credential types. For example, a custom credential type can inject an API token for a third-party web service into an environment variable, for your playbook or custom inventory script to consume.

Custom credentials support the following ways of injecting their authentication information:

- Environment variables
- Ansible extra variables
- File-based templating, which means generating **.ini** or **.conf** files that contain credential values

You can attach one SSH and multiple cloud credentials to a job template. Each cloud credential must be of a different type. Only one of each type of credential is permitted. Vault credentials and machine credentials are separate entities.

NOTE

- When creating a new credential type, you must avoid collisions in the **extra_vars**, **env**, and file namespaces.
- Environment variable or extra variable names must not start with **ANSIBLE_** because they are reserved.
- You must have System administrator (superuser) permissions to be able to create and edit a credential type (**CredentialType**) and to be able to view the **CredentialType.injection** field.

23.1. CONTENT SOURCING FROM COLLECTIONS

A "managed" credential type of **kind=galaxy** represents a content source for fetching collections defined in **requirements.yml** when project updates are run. Examples of content sources are galaxy.ansible.com, console.redhat.com, or on-premise automation hub. This new credential type represents a URL and (optional) authentication details necessary to construct the environment variables when a project update runs **ansible-galaxy collection install** as described in the Ansible documentation, Configuring the ansible-galaxy client. It has fields that map directly to the configuration options exposed to the Ansible Galaxy CLI, for example, per-server.

An endpoint in the API reflects an ordered list of these credentials at the Organization level:

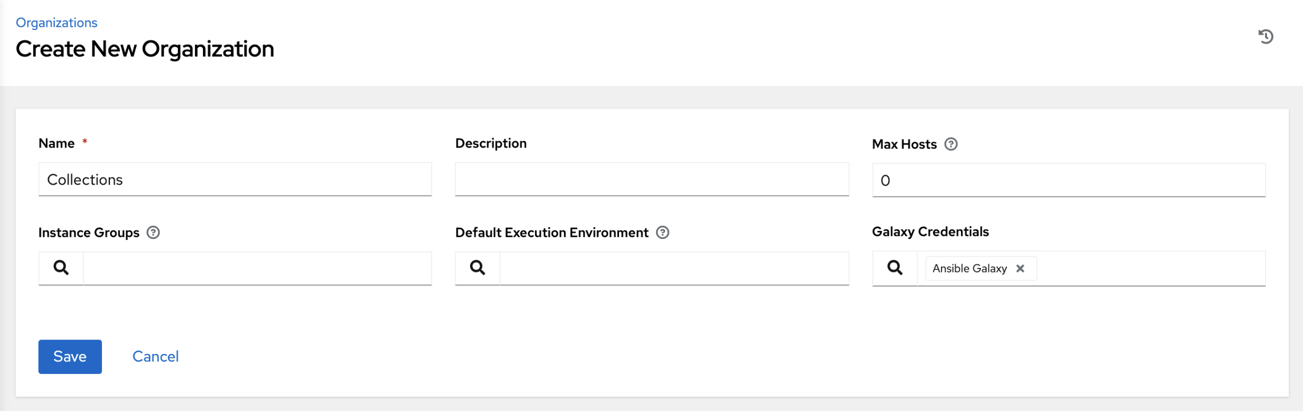
```
/api/v2/organizations/N/galaxy_credentials/
```

When installations of automation controller migrate existing Galaxy-oriented setting values, post-upgrade proper credentials are created and attached to every Organization. After upgrading to the latest version, every organization that existed before upgrade now has a list of one or more "Galaxy" credentials associated with it.

Additionally, post-upgrade, these settings are not visible (or editable) from the **/api/v2/settings/jobs/** endpoint.

Automation controller continues to fetch roles directly from public Galaxy even if **galaxy.ansible.com** is not the first credential in the list for the organization. The global Galaxy settings are no longer configured at the jobs level, but at the organization level in the user interface.

The organization's **Create organization** and **Edit organization** windows have an optional **Galaxy credentials** lookup field for credentials of **kind=galaxy**.



Organizations

Create New Organization

Name *	Description	Max Hosts
Collections		0
Instance Groups	Default Execution Environment	Galaxy Credentials
		Ansible Galaxy ×

It is important to specify the order of these credentials as order sets precedence for the sync and lookup of the content. For more information, see Creating an organization.

For more information about how to set up a project by using collections, see Using Collections with automation hub.

23.2. BACKWARDS-COMPATIBLE API CONSIDERATIONS

Support for version 2 of the API (**api/v2/**) means a one-to-many relationship for job templates to credentials (including multicloud support).

You can filter credentials the v2 API:

```
curl "https://controller.example.org/api/v2/credentials/?credential_type__namespace=aws"
```

In the V2 Credential Type model, the relationships are defined as follows:

Machine	SSH
Vault	Vault
Network	Sets environment variables, for example **ANSIBLE_NET_AUTHORIZE**
SCM	Source Control
Cloud	EC2, AWS
Cloud	Lots of others
Insights	Insights
Galaxy	galaxy.ansible.com, console.redhat.com

Machine	SSH
Galaxy	on-premise automation hub

23.3. CONTENT VERIFICATION

Automation controller uses GNU Privacy Guard (GPG) to verify content.

For more information, see The GNU Privacy Handbook .

23.4. GETTING STARTED WITH CREDENTIAL TYPES

Procedure

1. From the navigation panel, select **Automation Execution** → **Infrastructure** → **Credentials**. If no custom credential types have been created, the **Credential Types** page prompts you to add one.
   If credential types have been created, this page displays a list of existing and available Credential Types.

2. Select the name of a credential or the Edit ✎ icon to view more information about a credential type, .

3. On the **Details** tab, each credential type displays its own unique configurations in the **Input Configuration** field and the **Injector Configuration** field, if applicable. Both YAML and JSON formats are supported in the configuration fields.

23.5. CREATING A NEW CREDENTIAL TYPE

To create a new credential type:

Procedure

1. From the navigation panel, select **Automation Execution** → **Infrastructure** → **Credentials**.

2. In the **Credential Types** view, click **Create credential type**.

3. Enter the appropriate details in the **Name** and **Description** field.

   NOTE

   When creating a new credential type, do not use reserved variable names that start with **ANSIBLE_** for the **INPUT** and **INJECTOR** names and IDs, as they are invalid for custom credential types.

4. In the **Input configuration** field, specify an input schema that defines a set of ordered fields for that type. The format can be in YAML or JSON:
   YAML

```
fields:
  - type: string
    id: username
    label: Username
  - type: string
    id: password
    label: Password
    secret: true
required:
  - username
  - password
```

View more YAML examples at the YAML page.

JSON

```
{
"fields": [
  {
  "type": "string",
  "id": "username",
  "label": "Username"
  },
  {
  "secret": true,
  "type": "string",
  "id": "password",
  "label": "Password"
   }
  ],
 "required": ["username", "password"]
}
```

View more JSON examples at The JSON website.

The following configuration in JSON format shows each field and how they are used:

```
{
  "fields": [{
    "id": "api_token",    # required - a unique name used to reference the field value

    "label": "API Token", # required - a unique label for the field

    "help_text": "User-facing short text describing the field.",

    "type": ("string" | "boolean")   # defaults to 'string'

    "choices": ["A", "B", "C"]   # (only applicable to `type=string`)

    "format": "ssh_private_key"  # optional, can be used to enforce data format validity
                              for SSH private key data (only applicable to `type=string`)

    "secret": true,       # if true, the field value will be encrypted

    "multiline": false    # if true, the field should be rendered as multi-line for input entry
                       # (only applicable to `type=string`)
},{
    # field 2...
},{
    # field 3...
}],

"required": ["api_token"]   # optional; one or more fields can be marked as required
},
```

When **type=string**, fields can optionally specify multiple choice options:

```
{
  "fields": [{
      "id": "api_token",    # required - a unique name used to reference the field value
      "label": "API Token", # required - a unique label for the field
      "type": "string",
      "choices": ["A", "B", "C"]
  }]
},
```

5. In the **Injector configuration** field, enter environment variables or extra variables that specify the values a credential type can inject. The format can be in YAML or JSON (see examples in the previous step).
The following configuration in JSON format shows each field and how they are used:

```
{
  "file": {
      "template": "[mycloud]\ntoken={{ api_token }}"
  },
  "env": {
      "THIRD_PARTY_CLOUD_API_TOKEN": "{{ api_token }}"
  },
  "extra_vars": {
      "some_extra_var": "{{ username }}:{{ password }}"
  }
}
```

Credential Types can also generate temporary files to support **.ini** files or certificate or key data:

```
{
  "file": {
      "template": "[mycloud]\ntoken={{ api_token }}"
  },
  "env": {
      "MY_CLOUD_INI_FILE": "{{ tower.filename }}"
  }
}
```

In this example, automation controller writes a temporary file that has:

```
[mycloud]\ntoken=SOME_TOKEN_VALUE
```

The absolute file path to the generated file is stored in an environment variable named **MY_CLOUD_INI_FILE**.

The following is an example of referencing many files in a custom credential template:

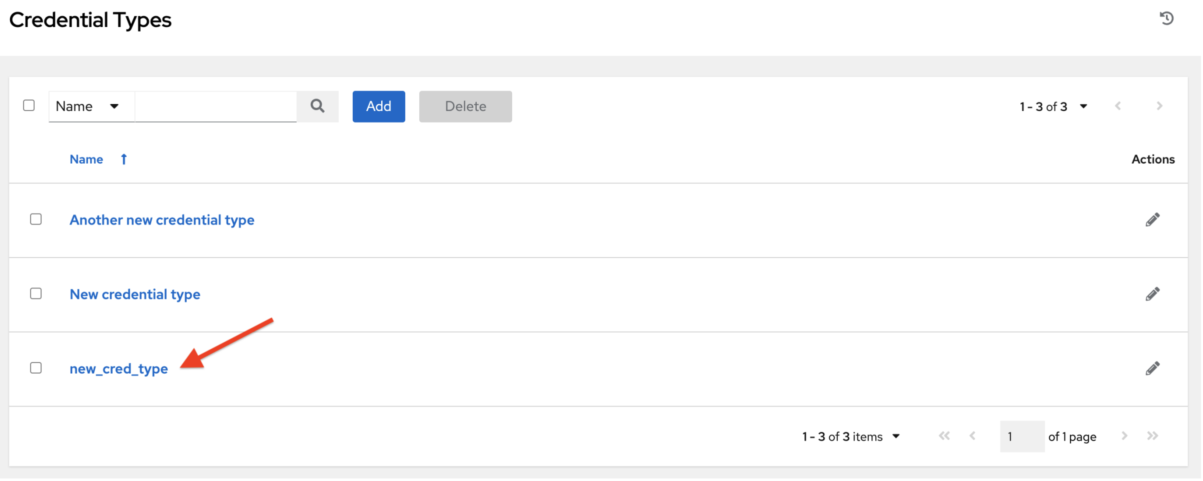
Inputs

```
{
  "fields": [{
    "id": "cert",
    "label": "Certificate",
    "type": "string"
  },{
    "id": "key",
    "label": "Key",
    "type": "string"
  }]
}
```

Injectors

```
{
  "file": {
    "template.cert_file": "[mycert]\n{{ cert }}",
    "template.key_file": "[mykey]\n{{ key }}"
},
"env": {
    "MY_CERT_INI_FILE": "{{ tower.filename.cert_file }}",
    "MY_KEY_INI_FILE": "{{ tower.filename.key_file }}"
}
}
```

6. Click **Create credential type**.
   Your newly created credential type is displayed on the list of credential types:



Credential Types

	Name	Actions
□	Another new credential type	✎
□	New credential type	✎
□	new_cred_type	✎

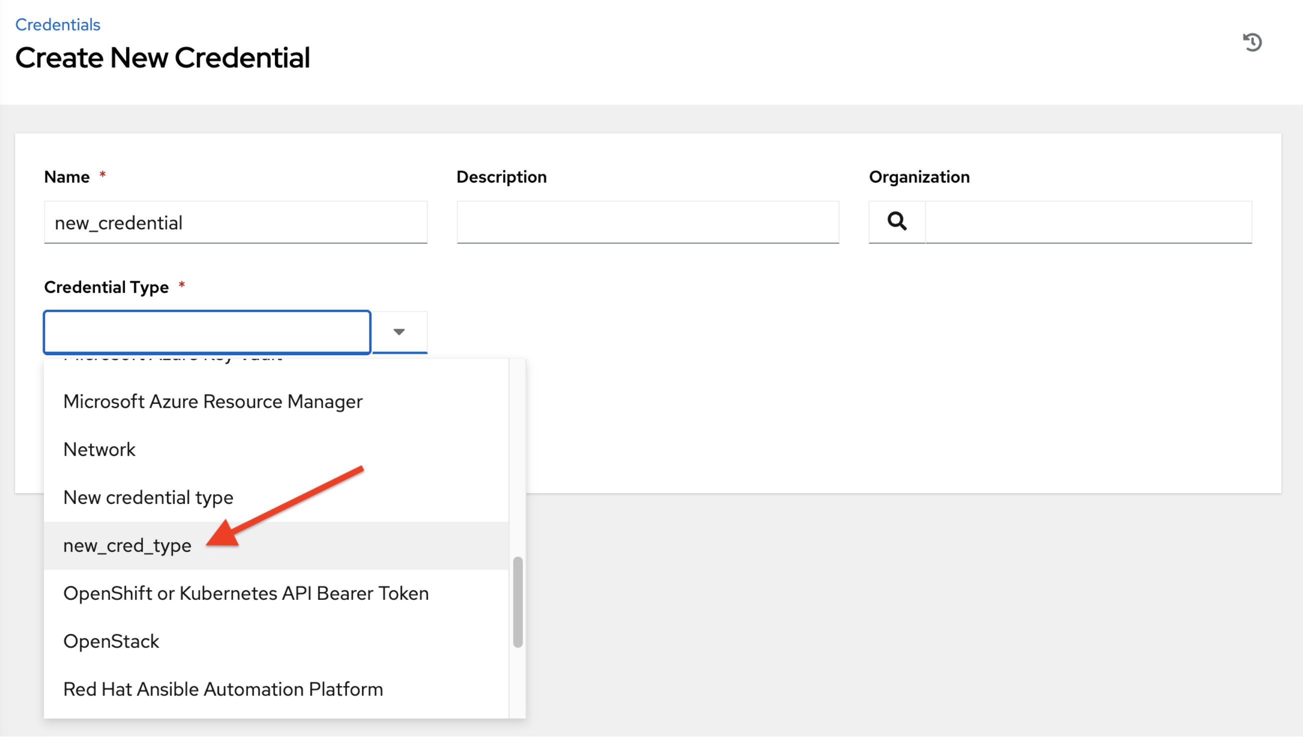
7. Click the Edit ✎ icon to modify the credential type options.

NOTE

In the **Edit** screen, you can modify the details or delete the credential. If the **Delete** option is disabled, this means that the credential type is being used by a credential, and you must delete the credential type from all the credentials that use it before you can delete it.

Verification

- Verify that the newly created credential type can be selected from the **Credential Type** selection window when creating a new credential:



Credentials

Create New Credential

Name *: new_credential

Description

Organization

Credential Type *

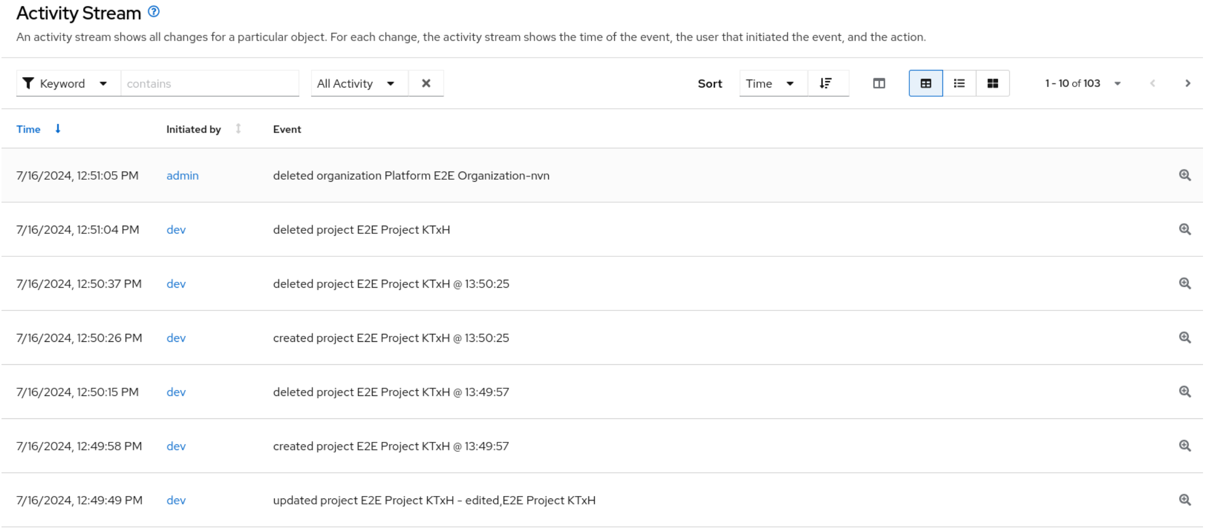
Microsoft Azure Resource Manager
Network
New credential type
new_cred_type
OpenShift or Kubernetes API Bearer Token
OpenStack
Red Hat Ansible Automation Platform

Additional resources

For information about how to create a new credential, see Creating a credential.

CHAPTER 24. ACTIVITY STREAM

- From the navigation panel, select **Automation Execution → Administration → Activity Stream**.



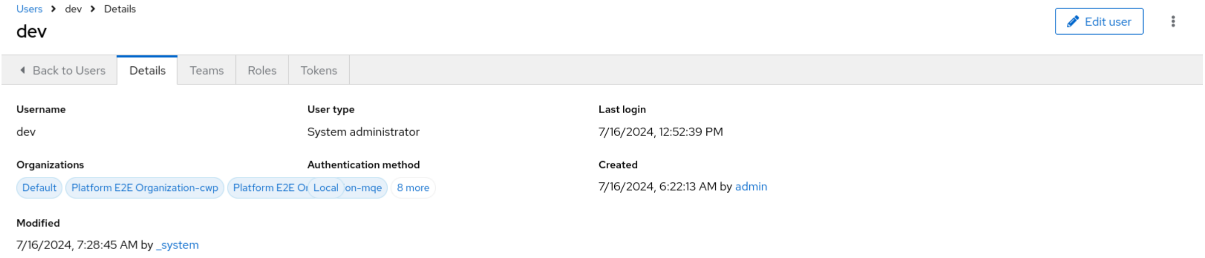
Activity Stream

An activity stream shows all changes for a particular object. For each change, the activity stream shows the time of the event, the user that initiated the event, and the action.

Time	Initiated by	Event
7/16/2024, 12:51:05 PM	admin	deleted organization Platform E2E Organization-nvn
7/16/2024, 12:51:04 PM	dev	deleted project E2E Project KTxH
7/16/2024, 12:50:37 PM	dev	deleted project E2E Project KTxH @ 13:50:25
7/16/2024, 12:50:26 PM	dev	created project E2E Project KTxH @ 13:50:25
7/16/2024, 12:50:15 PM	dev	deleted project E2E Project KTxH @ 13:49:57
7/16/2024, 12:49:58 PM	dev	created project E2E Project KTxH @ 13:49:57
7/16/2024, 12:49:49 PM	dev	updated project E2E Project KTxH - edited,E2E Project KTxH

An Activity Stream shows all changes for a particular object. For each change, the Activity Stream shows the time of the event, the user that initiated the event, and the action. The information displayed varies depending on the type of event.

- Click the 🔍 icon to display the event log for the change.



dev

Username dev	**User type** System administrator	**Last login** 7/16/2024, 12:52:39 PM
Organizations Default Platform E2E Organization-cwp Platform E2E Or	**Authentication method** Local on-mqe 8 more	**Created** 7/16/2024, 6:22:13 AM by admin
Modified 7/16/2024, 7:28:45 AM by _system		

You can filter the Activity Stream by the initiating user, by system (if it was system initiated), or by any related object, such as a credential, job template, or schedule. The Activity Stream shows the Activity Stream for the entire instance. Most pages permit viewing an activity stream filtered for that specific object.

You can view the activity stream on any page by clicking the **Activity Stream** ⟲ icon.

CHAPTER 25. NOTIFIERS

A Notification type such as Email, Slack or a Webhook, is an instance of a Notification Template, and has a name, description and configuration defined in the Notification template.

The following include examples of details needed to add a notification template:

- A username, password, server, and recipients are needed for an Email notification template
- The token and a list of channels are needed for a Slack notification template
- The URL and Headers are needed for a Webhook notification template

When a job fails, a notification is sent using the configuration that you define in the notification template.

The following shows the typical flow for the notification system:

- You create a notification template to the **REST API** at the **/api/v2/notification_templates endpoint**, either through the API or through the UI.
- You assign the notification template to any of the various objects that support it (all variants of job templates as well as organizations and projects) and at the appropriate trigger level for which you want the notification (started, success, or error). For example, you might want to assign a particular notification template to trigger when Job Template 1 fails. In this case, you associate the notification template with the job template at **/api/v2/job_templates/n/notification_templates_error** API endpoint.
- You can set notifications on job start and job end. Users and teams are also able to define their own notifications that can be attached to arbitrary jobs.

25.1. NOTIFICATION HIERARCHY

Notification templates inherit templates defined on parent objects, such as the following:

- Job templates use notification templates defined for them. Additionally, they can inherit notification templates from the project used by the job template, and from the organization that it is listed under.
- Project updates use notification templates defined on the project and inherit notification templates from the organization associated with it.
- Inventory updates use notification templates defined on the organization that it is listed under.
- Ad hoc commands use notification templates defined on the organization that the inventory is associated with.

25.2. NOTIFICATION WORKFLOW

When a job succeeds or fails, the error or success handler pulls a list of relevant notification templates using the procedure defined in the Notifiers section.

It then creates a notification object for each one, containing relevant details about the job and sends it to the destination. These include email addresses, slack channels, and SMS numbers.

These notification objects are available as related resources on job types (jobs, inventory updates, project updates), and also at **/api/v2/notifications**. You can also see what notifications have been sent from a notification template by examining its related resources.

If a notification fails, it does not impact the job associated with it or cause it to fail. The status of the notification can be viewed at its detail endpoint **/api/v2/notifications/<n>**.

25.3. CREATING A NOTIFICATION TEMPLATE

Use the following procedure to create a notification template.

Procedure

1. From the navigation panel, select **Automation Execution** → **Administration** → **Notifiers**.
2. Click **Add notifier**.
3. Complete the following fields:
    - **Name**: Enter the name of the notification.
    - **Description**: Enter a description for the notification. This field is optional.
    - **Organization**: Specify the organization that the notification belongs to.
    - **Type**: Choose a type of notification from the drop-down menu. For more information, see the Notification types section.
4. Click **Save notifier**.

25.4. NOTIFICATION TYPES

The following notification types are supported with automation controller:

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

Each notification type has its own configuration and behavioral semantics. You might need to test them in different ways. Additionally, you can customize each type of notification down to a specific detail or a set of criteria to trigger a notification.

Additional resources

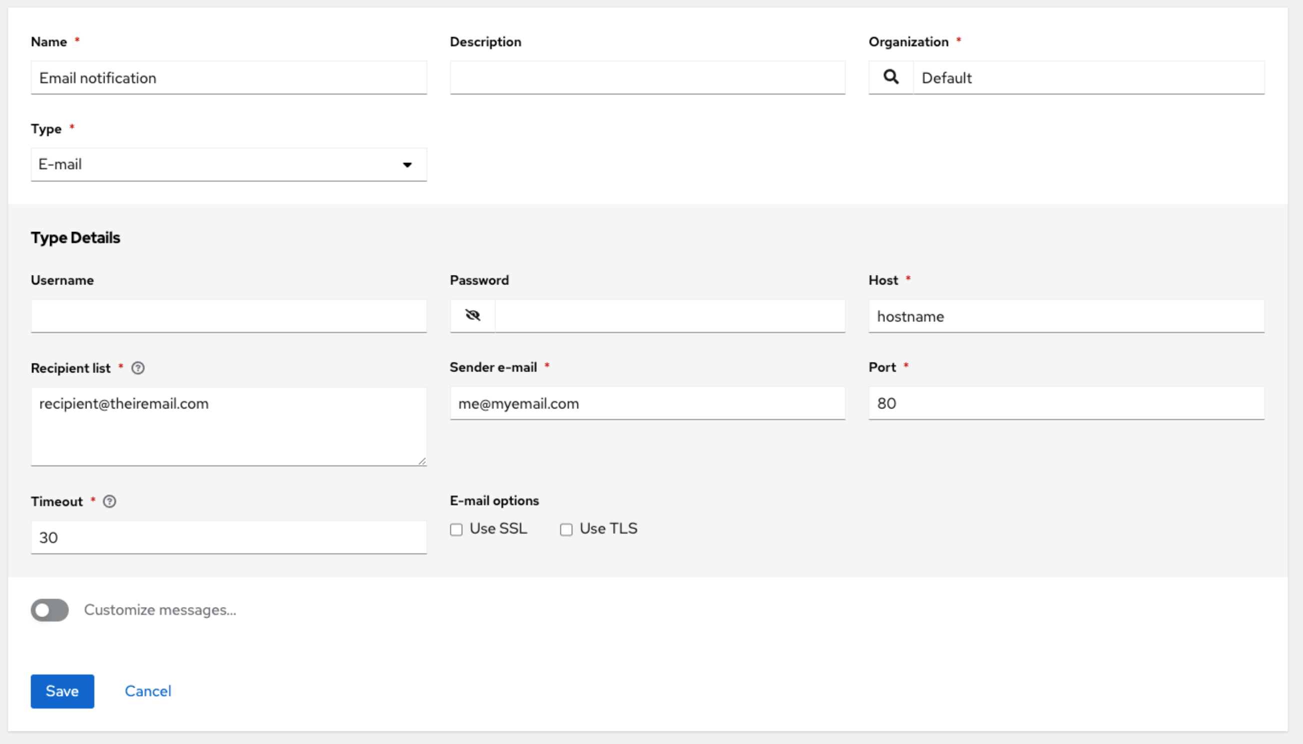
For more information on configuring custom notifications, see Create custom notifications. The following sections give further details on each type of notification.

25.4.1. Email

The email notification type supports a wide variety of SMTP servers and has support for SSL/TLS connections.

Provide the following details to set up an email notification:

- **Host**
- **Recipient list**
- **Sender e-mail**
- **Port**
- **Timeout** (in seconds): You can set this up to 120 seconds. This is the length of time that automation controller tries to connect to the email server before failure.



Name *: Email notification

Description:

Organization *: Default

Type *: E-mail

Type Details

Username:

Password:

Host *: hostname

Recipient list *: recipient@theiremail.com

Sender e-mail *: me@myemail.com

Port *: 80

Timeout *: 30

E-mail options: □ Use SSL □ Use TLS

25.4.2. Grafana

To integrate Grafana, you must first create an API key in the Grafana system. This is the token that is given to automation controller.

Provide the following details to set up a Grafana notification:

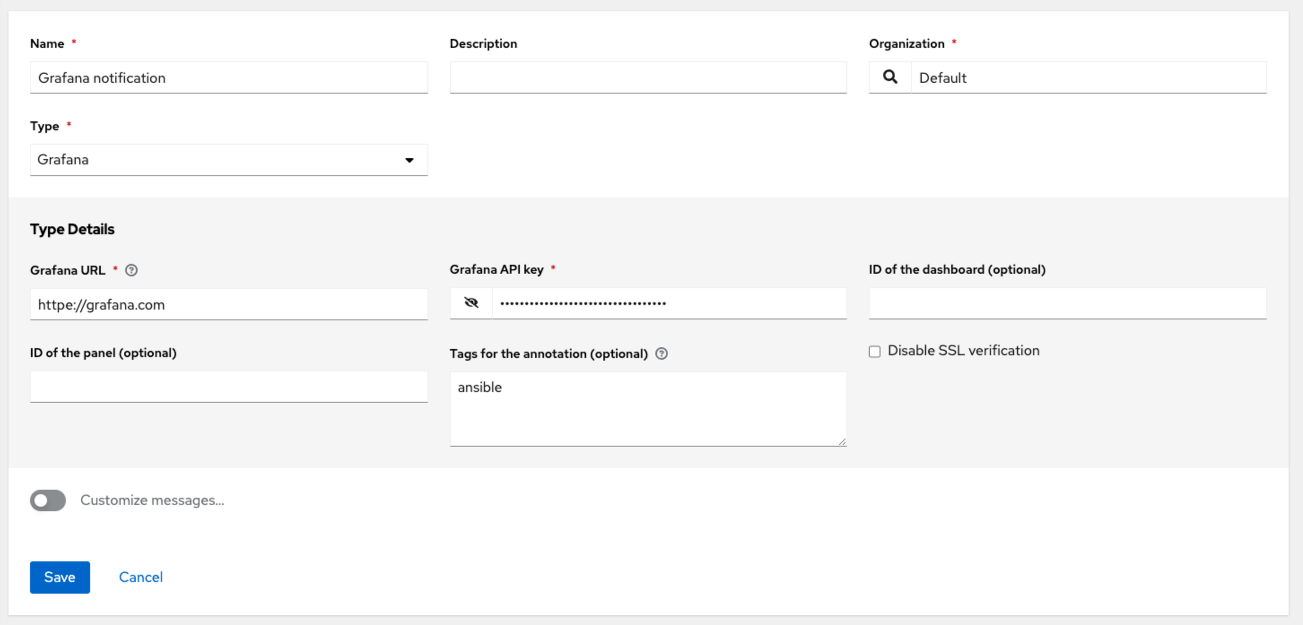
- **Grafana URL**: The URL of the Grafana API service, such as: http://yourcompany.grafana.com.
- **Grafana API key**: You must first create an API key in the Grafana system.

- Optional: **ID of the dashboard**: When you create an API key for the Grafana account, you can set up a dashboard with a unique ID.

- Optional: **ID of the panel**: If you added panels and graphs to your Grafana interface, you can give its ID here.

- Optional: **Tags for the annotation**: Enter keywords to identify the types of events of the notification that you are configuring.

- **Disable SSL verification**: SSL verification is on by default, but you can turn off verification of the authenticity of the target's certificate. Select this option to disable verification for environments that use internal or private CA's.



Name * Grafana notification

Description

Organization * Default

Type * Grafana

Type Details

Grafana URL * https://grafana.com

Grafana API key * ••••••••••••••••••••••••••••••••

ID of the dashboard (optional)

ID of the panel (optional)

Tags for the annotation (optional) ansible

□ Disable SSL verification

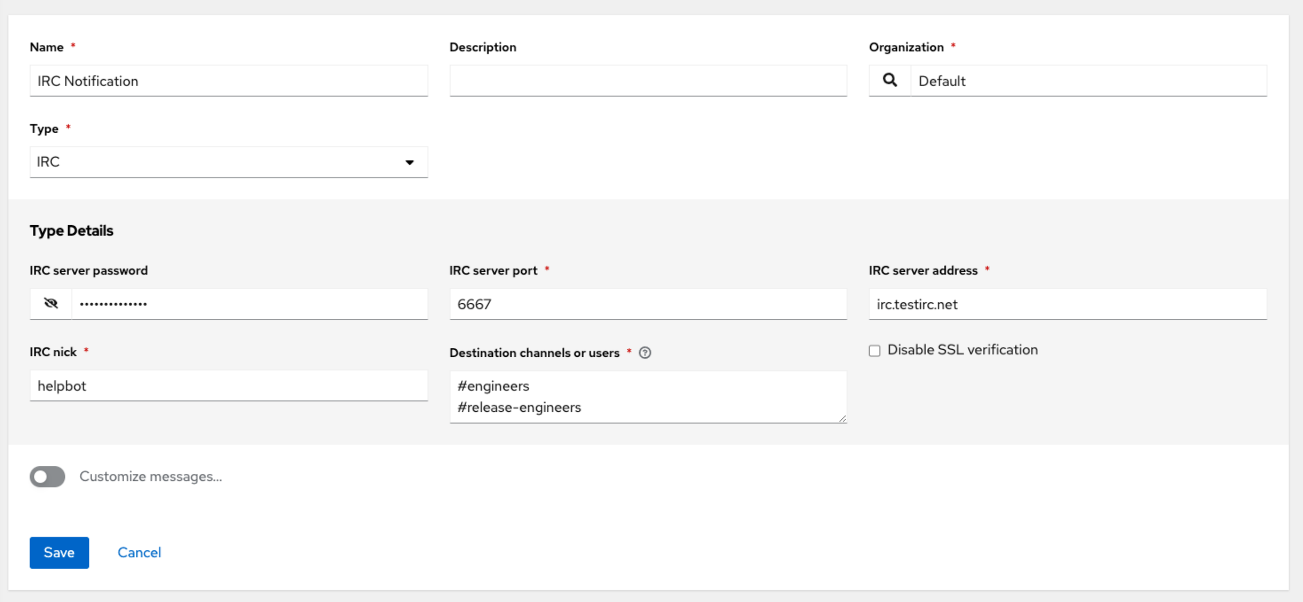
25.4.3. IRC

The IRC notification takes the form of an IRC bot that connects, delivers its messages to channels or individual users, and then disconnects. The notification bot also supports SSL authentication. The bot does not currently support Nickserv identification. If a channel or user does not exist or is not online then the notification fails. The failure scenario is reserved specifically for connectivity.

Provide the following details to set up an IRC notification:

- Optional: **IRC server password**: IRC servers can require a password to connect. If the server does not require one, leave it blank. **IRC Server Port**: The IRC server port. **IRC Server Address**: The host name or address of the IRC server. **IRC Nick**: The bot's nickname once it connects to the server. **Destination Channels or Users**: A list of users or channels to which the notification is sent.

- Optional: **Disable SSL verification**: Check if you want the bot to use SSL when connecting.



Name * IRC Notification

Description

Organization * Default

Type * IRC

Type Details

IRC server password ••••••••••••••

IRC server port * 6667

IRC server address * irc.testirc.net

IRC nick * helpbot

Destination channels or users *
#engineers
#release-engineers

□ Disable SSL verification

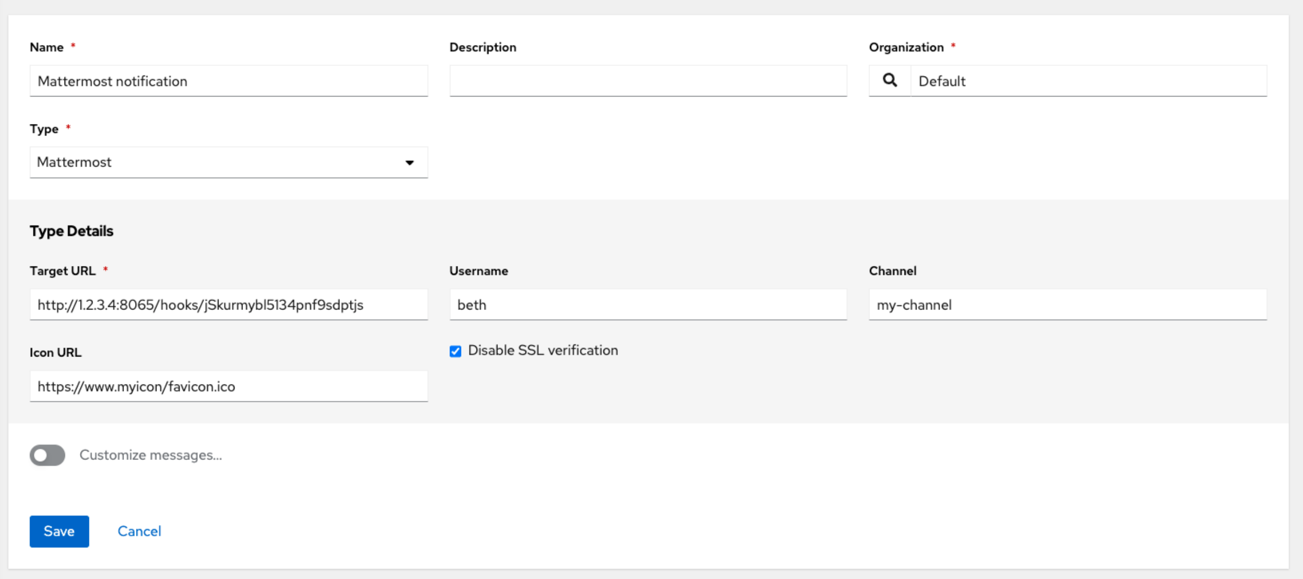
Customize messages...

25.4.4. Mattermost

The Mattermost notification type provides a simple interface to Mattermost's messaging and collaboration workspace.

Provide the following details to set up a Mattermost notification:

- **Target URL**: The full URL that is posted to.
- Optional: **Username**: Enter a username for the notification.
- Optional: **Channel**: Enter a channel for the notification.
- **Icon URL**: Specifies the icon to display for this notification.
- **Disable SSL verification**: Turns off verification of the authenticity of the target's certificate. Select this option to disable verification for environments that use internal or private CA's.



Name * Mattermost notification

Description

Organization * Default

Type * Mattermost

Type Details

Target URL * http://1.2.3.4:8065/hooks/jSkurmybl5134pnf9sdptjs

Username beth

Channel my-channel

Icon URL https://www.myicon/favicon.ico

☑ Disable SSL verification

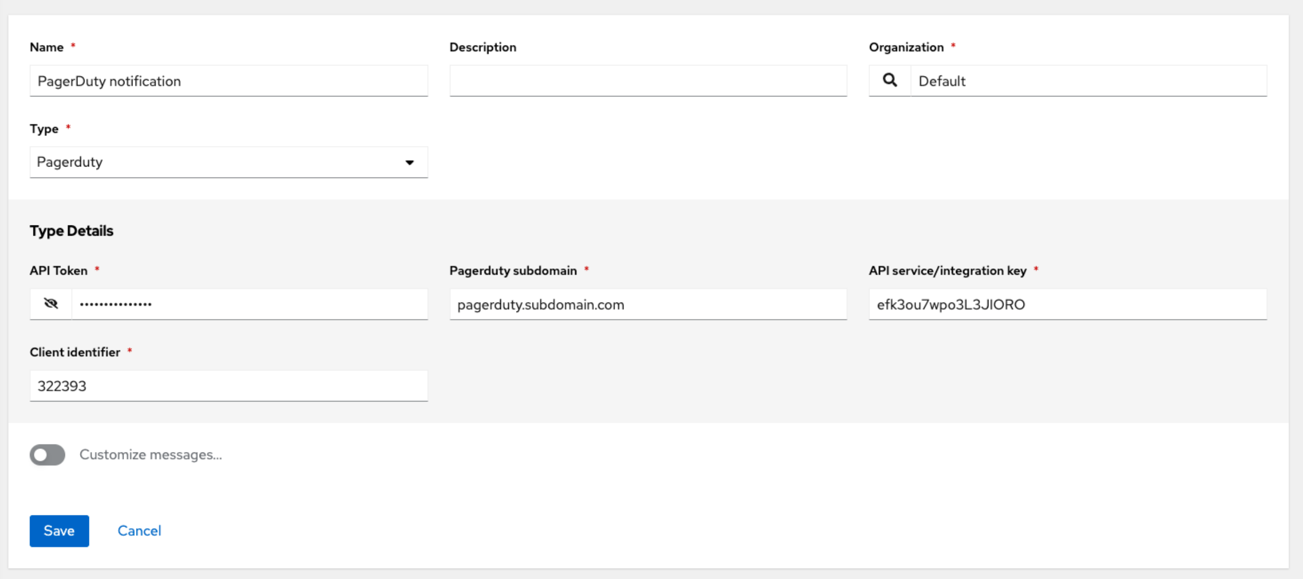
Customize messages...

25.4.5. Pagerduty

To integrate Pagerduty, you must first create an API key in the PagerDuty system. This is the token that is given to automation controller. Then create a **Service** which provides an **Integration Key** that is also given to automation controller.

Provide the following details to set up a Pagerduty notification:

- **API Token**: You must first create an API key in the Pagerduty system. This is the token that is given to automation controller.
- **PagerDuty subdomain**: When you sign up for the Pagerduty account, you receive a unique subdomain to communicate with. For example, if you signed up as "testuser", the web dashboard is at **testuser.pagerduty.com** and you give the API **testuser** as the subdomain, not the full domain.
- **API service/Integration Key**: Enter the API service/integration key created in Pagerduty.
- **Client Identifier**: This is sent along with the alert content to the Pagerduty service to help identify the service that is using the API key and service. This is helpful if multiple integrations are using the same API key and service.



Name *	Description	Organization *
PagerDuty notification		Default
Type *		
Pagerduty		

Type Details

API Token *	Pagerduty subdomain *	API service/integration key *
•••••••••••••••	pagerduty.subdomain.com	efk3ou7wpo3L3JIORO
Client identifier *		
322393		

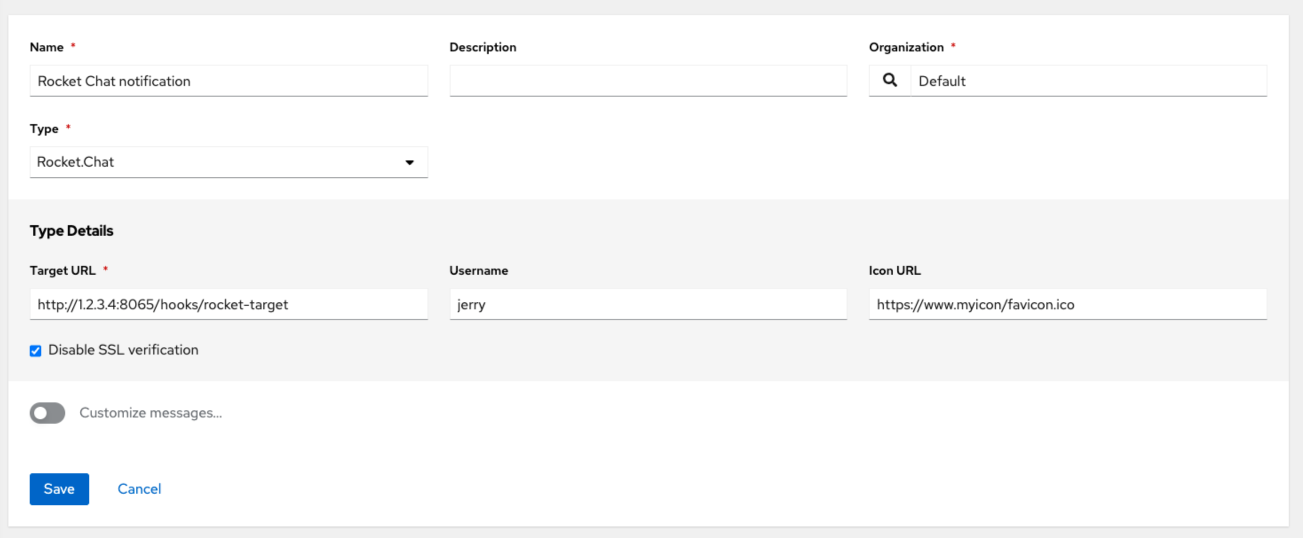
Customize messages...

25.4.6. Rocket.Chat

The Rocket.Chat notification type provides an interface to Rocket.Chat’s collaboration and communication platform.

Provide the following details to set up a Rocket.Chat notification:

- **Target URL**: The full URL that is **POSTed** to.
- Optional: **Username**: Enter a username.
- Optional: **Icon URL**: Specifies the icon to display for this notification
- **Disable SSL Verification**: Turns off verification of the authenticity of the target’s certificate. Select this option to disable verification for environments that use internal or private CA’s.



Name * Rocket Chat notification

Description

Organization * Default

Type * Rocket.Chat

Type Details

Target URL * http://1.2.3.4:8065/hooks/rocket-target

Username jerry

Icon URL https://www.myicon/favicon.ico

☑ Disable SSL verification

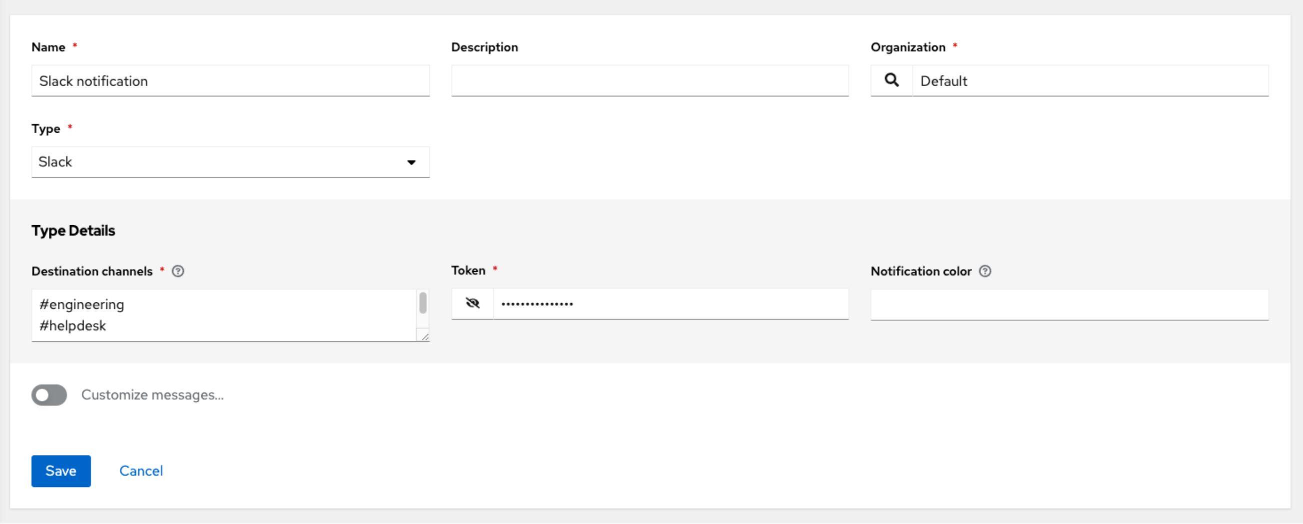
25.4.7. Slack

Slack is a collaborative team communication and messaging tool.

Provide the following details to set up a Slack notification:

- A Slack application. For more information, see the Quickstart page of the Slack documentation on how to create one.
- **Token**: A token. For more information, see Legacy bots and specific details on bot tokens on the Current token types documentation page.
- **Destination Channel**: One Slack channel per line. The pound symbol (#) is required for channels. To respond to or start a thread to a specific message add the parent message Id to the channel where the parent message Id is 16 digits. A dot (.) must be manually inserted after the 10th digit. For example, :#destination-channel, 1231257890.006423.
- **Notification color**: Specify a notification color. Acceptable colors are hex color code, for example: #3af or #789abc. When you have a bot or app set up, you must complete the following steps:

  1. Navigate to **Apps**.
  2. Click the newly-created app and then go to **Add features and functionality**, which enables you to configure incoming webhooks, bots, and permissions, as well as **Install your app to your workspace**.



Name * Slack notification

Description

Organization * Default

Type * Slack

Type Details

Destination channels * #engineering
#helpdesk

Token * •••••••••••••••

Notification color

25.4.8. Twilio

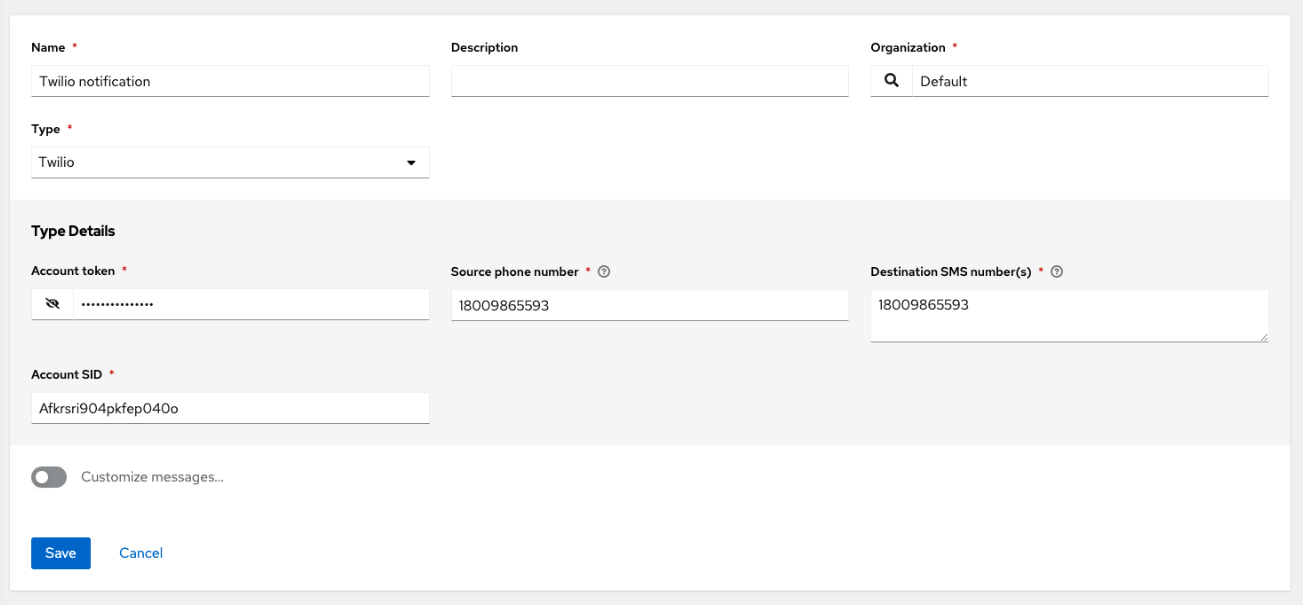
Twilio is a voice and SMS automation service. When you are signed in, you must create a phone number from which the messages are sent. You can then define a **Messaging Service** under **Programmable SMS** and associate the number you previously created with it.

You might need to verify this number or some other information before you are permitted to use it to send to any numbers. The **Messaging Service** does not require a status callback URL and it does not need the ability to process inbound messages.

Under your individual (or sub) account settings, you have API credentials. Twilio uses two credentials to determine which account an API request is coming from. The **Account SID**, which acts as a username, and the **Auth Token** which acts as a password.

Provide the following details to set up a Twilio notification:

- **Account SID**: Enter the account SID.
- **Account Token**: Enter the account token.
- **Source Phone Number**: Enter the number associated with the messaging service in the form of "+15556667777".
- **Destination SMS Numbers**: Enter the list of numbers you want to receive the SMS. It must be a 10 digit phone number.



Name * Twilio notification

Description

Organization * Default

Type * Twilio

Type Details

Account token * ···············

Source phone number * 18009865593

Destination SMS number(s) * 18009865593

Account SID * Afkrsri904pkfep040o

Customize messages...

25.4.9. Webhook

The webhook notification type provides a simple interface for sending **POSTs** to a predefined web service. Automation controller **POSTs** to this address using application and JSON content type with the data payload containing the relevant details in JSON format. Some web service APIs expect HTTP requests to be in a certain format with certain fields.

Configure the webhook notification with the following:

- Configure the HTTP method, using **POST** or **PUT**.
- The body of the outgoing request.
- Configure authentication, using basic auth.

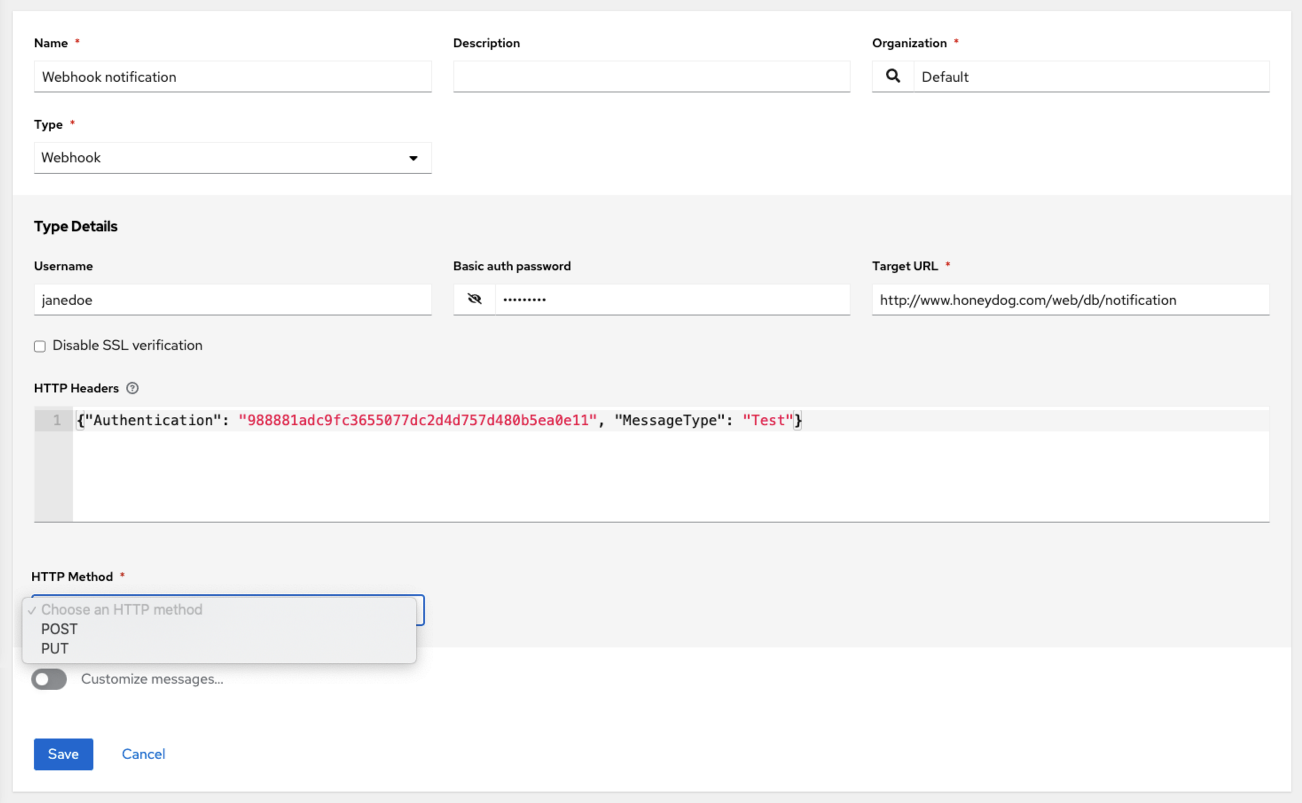
Provide the following details to set up a webhook notification:

- Optional: **Username**: Enter a username.
- Optional: **Basic auth password**
- **Target URL**: Enter the full URL to which the webhook notification is **PUT** or **POSTed**.
- **HTTP Headers**: Enter Headers in JSON format where the keys and values are strings. For example:

```
{"Authentication": "988881adc9fc3655077dc2d4d757d480b5ea0e11", "MessageType": "Test"}`.
```

- **Disable SSL Verification**: SSL verification is on by default, but you can choose to turn off verification of the authenticity of the target's certificate. Select this option to disable verification for environments that use internal or private CA's.
- **HTTP Method**: Select the method for your webhook:
- **POST**: Creates a new resource. It also acts as a catch-all for operations that do not fit into the other categories. It is likely that you need to **POST** unless you know your webhook service expects a **PUT**.

- **PUT**: Updates a specific resource (by an identifier) or a collection of resources. You can also use **PUT** to create a specific resource if the resource identifier is known beforehand.



Name * Webhook notification

Description

Organization * Default

Type * Webhook

Type Details

Username janedoe

Basic auth password ·········

Target URL * http://www.honeydog.com/web/db/notification

□ Disable SSL verification

HTTP Headers

```
{"Authentication": "988881adc9fc3655077dc2d4d757d480b5ea0e11", "MessageType": "Test"}
```

HTTP Method *

✓ Choose an HTTP method
POST
PUT

Customize messages...

25.4.9.1. Webhook payloads

The following data is sent by automation controller at the webhook endpoint:

```
job id
name
url
created_by
started
finished
status
traceback
inventory
project
playbook
credential
limit
extra_vars
hosts
http method
```

The following is an example of a **started** notification through a webhook message as returned by automation controller:

```
{"id": 38, "name": "Demo Job Template", "url": "https://host/#/jobs/playbook/38", "created_by":
"bianca", "started":
"2020-07-28T19:57:07.888193+00:00", "finished": null, "status": "running", "traceback": "", "inventory":
"Demo Inventory",
"project": "Demo Project", "playbook": "hello_world.yml", "credential": "Demo Credential", "limit": "",
"extra_vars": "{}",
"hosts": {}}POST / HTTP/1.1
```

The following data is returned by automation controller at the webhook endpoint for a **success**/**fail** status:

```
job id
name
url
created_by
started
finished
status
traceback
inventory
project
playbook
credential
limit
extra_vars
hosts
```

The following is an example of a **success**/**fail** notification as returned by automation controller through a webhook message:

```
{"id": 46, "name": "AWX-Collection-tests-awx_job_wait-long_running-XVFBGRSAvUUIrYKn", "url":
"https://host/#/jobs/playbook/46",
"created_by": "bianca", "started": "2020-07-28T20:43:36.966686+00:00", "finished": "2020-07-
28T20:43:44.936072+00:00", "status": "failed",
"traceback": "", "inventory": "Demo Inventory", "project": "AWX-Collection-tests-awx_job_wait-
long_running-JJSlglnwtsRJyQmw", "playbook":
"fail.yml", "credential": null, "limit": "", "extra_vars": "{\"sleep_interval\": 300}", "hosts": {"localhost":
{"failed": true, "changed": 0,
"dark": 0, "failures": 1, "ok": 1, "processed": 1, "skipped": 0, "rescued": 0, "ignored": 0}}}
```

25.5. CREATING CUSTOM NOTIFICATIONS

You can customize the text content of each Notification type on the notification form.

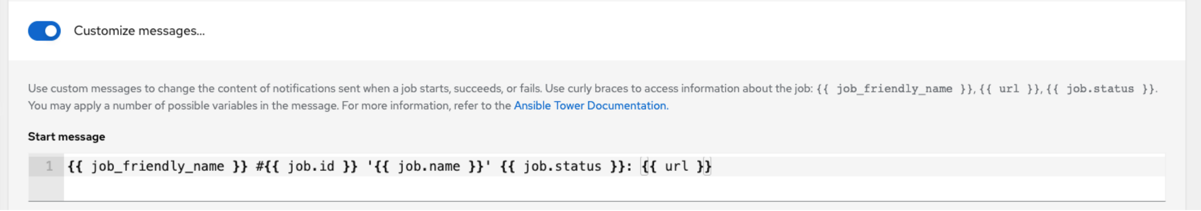
Procedure

1. From the navigation panel, select **Automation Execution** → **Administration** → **Notifiers**.

2. Click **Create notifier**.

3. Choose a notification type from the **Type** list.

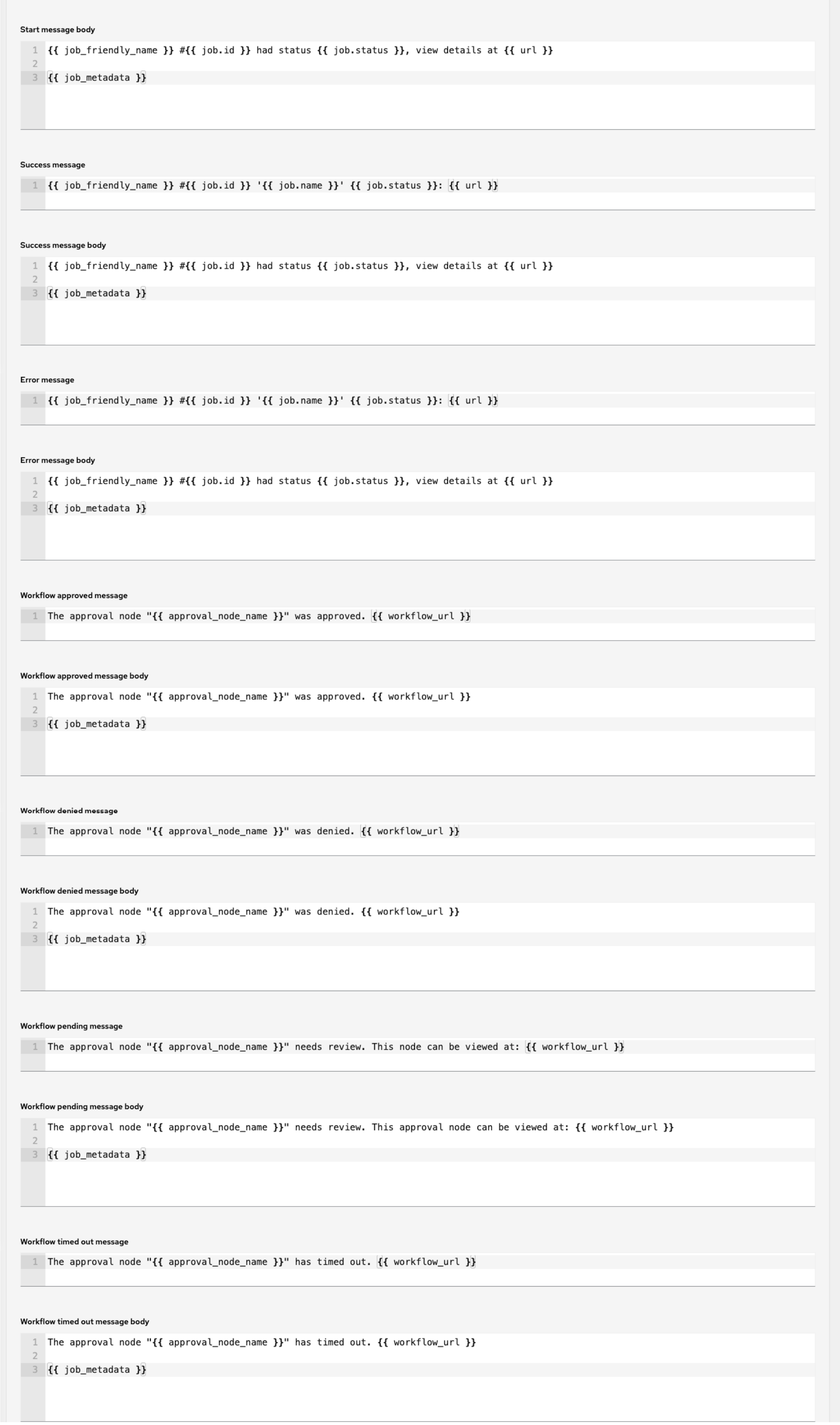
4. Enable **Customize messages** by using the toggle.



Use custom messages to change the content of notifications sent when a job starts, succeeds, or fails. Use curly braces to access information about the job: {{ job_friendly_name }}, {{ url }}, {{ job.status }}. You may apply a number of possible variables in the message. For more information, refer to the Ansible Tower Documentation.

Start message

```
{{ job_friendly_name }} #{{ job.id }} '{{ job.name }}' {{ job.status }}: {{ url }}
```



Start message body

```
{{ job_friendly_name }} #{{ job.id }} had status {{ job.status }}, view details at {{ url }}

{{ job_metadata }}
```

Success message

```
{{ job_friendly_name }} #{{ job.id }} '{{ job.name }}' {{ job.status }}: {{ url }}
```

Success message body

```
{{ job_friendly_name }} #{{ job.id }} had status {{ job.status }}, view details at {{ url }}

{{ job_metadata }}
```

Error message

```
{{ job_friendly_name }} #{{ job.id }} '{{ job.name }}' {{ job.status }}: {{ url }}
```

Error message body

```
{{ job_friendly_name }} #{{ job.id }} had status {{ job.status }}, view details at {{ url }}

{{ job_metadata }}
```

Workflow approved message

```
The approval node "{{ approval_node_name }}" was approved. {{ workflow_url }}
```

Workflow approved message body

```
The approval node "{{ approval_node_name }}" was approved. {{ workflow_url }}

{{ job_metadata }}
```

Workflow denied message

```
The approval node "{{ approval_node_name }}" was denied. {{ workflow_url }}
```

Workflow denied message body

```
The approval node "{{ approval_node_name }}" was denied. {{ workflow_url }}

{{ job_metadata }}
```

Workflow pending message

```
The approval node "{{ approval_node_name }}" needs review. This node can be viewed at: {{ workflow_url }}
```

Workflow pending message body

```
The approval node "{{ approval_node_name }}" needs review. This approval node can be viewed at: {{ workflow_url }}

{{ job_metadata }}
```

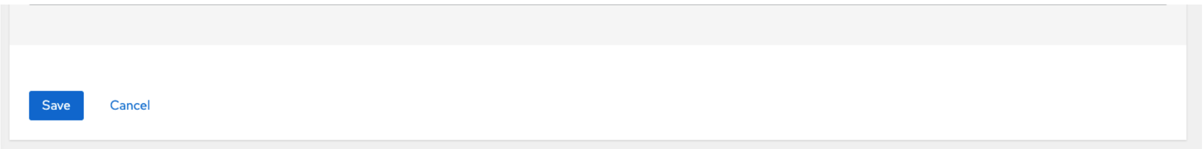
Workflow timed out message

```
The approval node "{{ approval_node_name }}" has timed out. {{ workflow_url }}
```

Workflow timed out message body

```
The approval node "{{ approval_node_name }}" has timed out. {{ workflow_url }}

{{ job_metadata }}
```



5. You can provide a custom message for various job events, such as the following:

    - **Start message body**
    - **success message body**
    - **Error message body**
    - **Workflow approved body**
    - **Workflow denied message body**
    - **Workflow pending message body**
    - **Workflow timed out message body**

The message forms vary depending on the type of notification that you are configuring. For example, messages for Email and PagerDuty notifications appear to be a typical email, with a body and a subject, in which case, automation controller displays the fields as **Message** and **Message Body**. Other notification types only expect a **Message** for each type of event.

The **Message** fields are pre-populated with a template containing a top-level variable, **job** coupled with an attribute, such as **id** or **name**. Templates are enclosed in curly brackets and can draw from a fixed set of fields provided by automation controller, shown in the pre-populated message fields:

This pre-populated field suggests commonly displayed messages to a recipient who is notified of an event. You can customize these messages with different criteria by adding your own attributes for the job as needed. Custom notification messages are rendered using Jinja; the same templating engine used by Ansible playbooks.

Messages and message bodies have different types of content, as the following points outline:

- Messages are always just strings, one-liners only. New lines are not supported.
- Message bodies are either a dictionary or a block of text:
    - The message body for Webhooks and PagerDuty uses dictionary definitions. The default message body for these is **{{ job_metadata }}**, you can either leave that as it is or provide your own dictionary.
    - The message body for email uses a block of text or a multi-line string. The default message body is:

        > {{ job_friendly_name }} #{{ job.id }} had status {{ job.status }}, view details at {{ url }} {{ job_metadata }}

        You can edit this text leaving **{{ job_metadata }}** in, or drop **{{ job_metadata }}**. Since the body is a block of text, it can be any string you want. **{{ job_metadata }}** is rendered as a dictionary containing fields that describe the job being executed. In all cases, **{{ job_metadata }}** includes the following fields:

        - **id**

- **name**
- **url**
- **created_by**
- **started**
- **finished**
- **status**
- **traceback**

NOTE

You cannot query individual fields within **{{ job_metadata }}**. When you use **{{ job_metadata }}** in a notification template, all data is returned.

The resulting dictionary looks like the following:

```
{"id": 18,
 "name": "Project - Space Procedures",
 "url": "https://host/#/jobs/project/18",
 "created_by": "admin",
 "started": "2019-10-26T00:20:45.139356+00:00",
 "finished": "2019-10-26T00:20:55.769713+00:00",
 "status": "successful",
 "traceback": ""
}
```

If **{{ job_metadata }}** is rendered in a job, it includes the following additional fields:

- **inventory**
- **project**
- **playbook**
- **credential**
- **limit**
- **extra_vars**
- **hosts**
  The resulting dictionary is similar to the following:

```
{"id": 12,
 "name": "JobTemplate - Launch Rockets",
 "url": "https://host/#/jobs/playbook/12",
 "created_by": "admin",
 "started": "2019-10-26T00:02:07.943774+00:00",
 "finished": null,
 "status": "running",
 "traceback": "",
 "inventory": "Inventory - Fleet",
 "project": "Project - Space Procedures",
 "playbook": "launch.yml",
 "credential": "Credential - Mission Control",
 "limit": "",
 "extra_vars": "{}",
 "hosts": {}
}
```

If **{{ job_metadata }}** is rendered in a workflow job, it includes the following additional field:

- **body** (This enumerates the nodes in the workflow job and includes a description of the job associated with each node)
  The resulting dictionary is similar to the following:

```
{"id": 14,
 "name": "Workflow Job Template - Launch Mars Mission",
 "url": "https://host/#/workflows/14",
 "created_by": "admin",
 "started": "2019-10-26T00:11:04.554468+00:00",
 "finished": "2019-10-26T00:11:24.249899+00:00",
 "status": "successful",
 "traceback": "",
 "body": "Workflow job summary:

        node #1 spawns job #15, \"Assemble Fleet JT\", which finished with status
successful.
        node #2 spawns job #16, \"Mission Start approval node\", which finished with
status successful.\n
        node #3 spawns job #17, \"Deploy Fleet\", which finished with status
successful."
}
```

If you create a notification template that uses invalid syntax or references unusable fields, an error message displays indicating the nature of the error. If you delete a notification's custom message, the default message is shown in its place.

IMPORTANT

If you save the notifications template without editing the custom message (or edit and revert back to the default values), the **Details** screen assumes the defaults and does not display the custom message tables. If you edit and save any of the values, the entire table displays in the **Details** screen.

Additional resources

- For more information, see Using variables with Jinja2 in the Ansible documentation.
- Automation controller requires valid syntax to retrieve the correct data to display the messages.

For a list of supported attributes and the proper syntax construction, see the Supported Attributes for Custom Notifications section.

25.6. ENABLE AND DISABLE NOTIFICATIONS

You can set up notifications to notify you when a specific job starts, as well as on the success or failure at the end of the job run. Note the following behaviors:

- If a workflow job template has notification on start enabled, and a job template within that workflow also has notification on start enabled, you receive notifications for both.
- You can enable notifications to run on many job templates within a workflow job template.
- You can enable notifications to run on a sliced job template start and each slice generates a notification.
- When you enable a notification to run on job start, and that notification gets deleted, the job template continues to run, but results in an error message.

You can enable notifications on job start, job success, and job failure, or a combination of these, from the **Notifications** tab of the **Details** page for the following resources:

- Job Templates
- Workflow Templates
- Projects

For workflow templates that have approval nodes, in addition to **Start**, **Success**, and **Failure**, you can enable or disable certain approval-related events:

Additional resources

For more information on working with these types of nodes, see Approval nodes.

25.7. CONFIGURE THE HOST HOSTNAME FOR NOTIFICATIONS

In System settings, you can replace the default value in the **Base URL of the service**field with your preferred hostname to change the notification hostname.

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

25.7.1. Resetting TOWER_URL_BASE

Automation controller determines how the base URL (**TOWER_URL_BASE**) is defined by looking at an incoming request and setting the server address based on that incoming request.

Automation controller takes settings values from the database first. If no settings values are found, it uses the values from the settings files. If you post a license by navigating to the automation controller host's IP address, the posted license is written to the settings entry in the database.

Use the following procedure to reset **TOWER_URL_BASE** if the wrong address has been picked up:

Procedure

1. From the navigation panel, select **Settings → System**.
2. Click **Edit**.

3. Enter the address in the **Base URL of the service** field for the DNS entry you want to appear in notifications.

25.8. NOTIFICATIONS API

Use the **started**, **success**, or **error** endpoints:

```
/api/v2/organizations/N/notification_templates_started/
/api/v2/organizations/N/notification_templates_success/
/api/v2/organizations/N/notification_templates_error/
```

Additionally, the **../../../N/notification_templates_started** endpoints have **GET** and **POST** actions for:

- Organizations
- Projects
- Inventory Sources
- Job Templates
- System Job Templates
- Workflow Job Templates

CHAPTER 26. SUPPORTED ATTRIBUTES FOR CUSTOM NOTIFICATIONS

Learn about the list of supported job attributes and the proper syntax for constructing the message text for notifications.

The following are the supported job attributes:

- **allow_simultaneous** - (boolean) Indicates if multiple jobs can run simultaneously from the job template associated with this job.
- **controller_node** - (string) The instance that manages the isolated execution environment.
- **created** - (datetime) The timestamp when this job was created.
- **custom_virtualenv** - (string) The custom virtual environment used to execute the job.
- **description** - (string) An optional description of the job.
- **diff_mode** - (boolean) If enabled, textual changes made to any templated files on the host are shown in the standard output.
- **elapsed** - (decimal) The elapsed time in seconds that the job has run.
- **execution_node** - (string) The node that the job executes on.
- **failed** - (boolean) True if the job failed.
- **finished** - (datetime) The date and time the job finished execution.
- **force_handlers** - (boolean) When handlers are forced, they run when notified even if a task fails on that host. Note that some conditions, such as unreachable hosts can still prevent handlers from running.
- **forks** - (int) The number of forks requested for this job.
- **id** - (int) The database ID for this job.
- **job_explanation** - (string) The status field to indicate the state of the job if it was not able to run and capture **stdout**.
- **job_slice_count** - (integer) If run as part of a sliced job, this is the total number of slices (if 1, job is not part of a sliced job).
- **job_slice_number** - (integer) If run as part of a sliced job, this is the ID of the inventory slice operated on (if not part of a sliced job, attribute is not used).
- **job_tags** - (string) Only tasks with specified tags execute.
- **job_type** - (choice) This can be **run**, **check**, or **scan**.
- **launch_type** - (choice) This can be **manual**, **relaunch**, **callback**, **scheduled**, **dependency**, **workflow**, **sync**, or **scm**.
- **limit** - (string) The playbook execution limited to this set of hosts, if specified.
- **modified** - (datetime) The timestamp when this job was last modified.

- **name** - (string) The name of this job.
- **playbook** - (string) The playbook executed.
- **scm_revision** - (string) The scm revision from the project used for this job, if available.
- **skip_tags** - (string) The playbook execution skips over this set of tags, if specified.
- **start_at_task** - (string) The playbook execution begins at the task matching this name, if specified.
- **started**- (datetime) The date and time the job was queued for starting.
- **status** - (choice) This can be **new**, **pending**, **waiting**, **running**, **successful**, **failed**, **error**, or **canceled**.
- **timeout** - (int) The amount of time, in seconds, to run before the task is canceled.
- **type** - (choice) The data type for this job.
- **url** - (string) The URL for this job.
- **use_fact_cache** - (boolean) If enabled for the job, automation controller acts as an Ansible Fact Cache Plugin at the end of a playbook run to the database and caches facts for use by Ansible.
- **verbosity** - (choice) 0 through 5 (corresponding to Normal through WinRM Debug).
  - **host_status_counts** (The count of hosts uniquely assigned to each status)
    - **skipped** (integer)
    - **ok** (integer)
    - **changed** (integer)
    - **failures** (integer)
    - **dark** (integer)
    - **processed** (integer)
    - **rescued** (integer)
    - **ignored** (integer)
    - **failed** (boolean)
  - **summary_fields**:
    - **inventory**
      - **id** - (integer) The database ID for the inventory.
      - **name** - (string) The name of the inventory.
      - **description** - (string) An optional description of the inventory.

    - **has_active_failures**- (boolean) (deprecated) flag indicating whether any hosts in this inventory have failed.
    - **total_hosts** - (deprecated) (int) The total number of hosts in this inventory.
    - **hosts_with_active_failures** - (deprecated) (int) The number of hosts in this inventory with active failures.
    - **total_groups** - (deprecated) (int) The total number of groups in this inventory.
    - **groups_with_active_failures** - (deprecated) (int) The number of hosts in this inventory with active failures.
    - **has_inventory_sources** - (deprecated) (boolean) The flag indicating whether this inventory has external inventory sources.
    - **total_inventory_sources** - (int) The total number of external inventory sources configured within this inventory.
    - **inventory_sources_with_failures** - (int) The number of external inventory sources in this inventory with failures.
    - **organization_id** - (id) The organization containing this inventory.
    - **kind** - (choice) (empty string) (indicating hosts have direct link with inventory) or **smart**

  - **project**
    - **id** - (int) The database ID for the project.
    - **name** - (string) The name of the project.
    - **description** (string) An optional description of the project.
    - **status** - (choices) One of **new**, **pending**, **waiting**, **running**, **successful**, **failed**, **error**, **canceled**, **never updated**, **ok**, or **missing**.
    - **scm_type** (choice) One of (empty string), **git**, **hg**, **svn**, **insights**.

  - **job_template**
    - **id** - (int) The database ID for the job template.
    - **description** - (string) The optional description of the project.
    - **status** - (choices) One of **new**, **pending**, **waiting**, **running**, **successful**, **failed**, **error**, **canceled**, **never updated**, **ok**, or **missing**.

  - **job_template**
    - **id**- (int) The database ID for the job template.
    - **name**- (string) The name of the job template.
    - **description**- (string) An optional description for the job template.

  - **unified_job_template**

  - **id** - (int) The database ID for the unified job template.
  - **name** - (string) The name of the unified job template.
  - **description** - (string) An optional description for the unified job template.
  - **unified_job_type** - (choice) The unified job type, such as **job**, **workflow_job**, or **project_update**.

- **instance_group**
  - **id** - (int) The database ID for the instance group.
  - **name**- (string) The name of the instance group.
- **created_by**
  - **id** - (int) The database ID of the user that launched the operation.
  - **username** - (string) The username that launched the operation.
  - **first_name** - (string) The first name.
  - **last_name** - (string) The last name.
- **labels**
  - **count** - (int) The number of labels.
  - **results** - The list of dictionaries representing labels. For example, {"id": 5, "name": "database jobs"}.

You can reference information about a job in a custom notification message using grouped curly brackets {{ }}. Specific job attributes are accessed using dotted notation, for example, {{ job.summary_fields.inventory.name }}. You can add any characters used in front or around the braces, or plain text, for clarification, such as "#" for job ID and single-quotes to denote some descriptor. Custom messages can include a number of variables throughout the message:

> {{ job_friendly_name }} {{ job.id }} ran on {{ job.execution_node }} in {{ job.elapsed }} seconds.

The following are additional variables that can be added to the template:

- **approval_node_name** - (string) The approval node name.
- **approval_status** - (choice) One of **approved**, **denied**, and **timed_out**.
- **url** - (string) The URL of the job for which the notification is emitted (this applies to **start**, **success**, **fail**, and **approval notifications**).
- **workflow_url** - (string) The URL to the relevant approval node. This allows the notification recipient to go to the relevant workflow job page to examine the situation. For example, **This node can be viewed at: {{workflow_url }}**. In cases of approval-related notifications, both **url** and **workflow_url** are the same.
- **job_friendly_name** - (string) The friendly name of the job.
- **job_metadata** - (string) The job metadata as a JSON string, for example:

```
{'url': 'https://automationcontroller.example.com/$/jobs/playbook/13',
 'traceback': '',
 'status': 'running',
 'started': '2019-08-07T21:46:38.362630+00:00',
 'project': 'Stub project',
 'playbook': 'ping.yml',
 'name': 'Stub Job Template',
 'limit': '',
 'inventory': 'Stub Inventory',
 'id': 42,
 'hosts': {},
 'friendly_name': 'Job',
 'finished': False,
 'credential': 'Stub credential',
 'created_by': 'admin'}
```

CHAPTER 27. WORKING WITH WEBHOOKS

A Webhook enables you to execute specified commands between applications over the web. Automation controller currently provides webhook integration with GitHub and GitLab.

Set up a webhook using the following services:

- Setting up a GitHub webhook
- Setting up a GitLab webhook
- Viewing a payload output

The webhook post-status-back functionality for GitHub and GitLab is designed to work only under certain CI events. Receiving another kind of event results in messages such as the following in the service log:

awx.main.models.mixins Webhook event did not have a status API endpoint associated, skipping.

27.1. SETTING UP A GITHUB WEBHOOK

Automation controller has the ability to run jobs based on a triggered webhook event coming in. Job status information (pending, error, success) can be sent back only for pull request events. If you do not need automation controller to post job statuses back to the webhook service, go directly to step 3.

Procedure

1. Generate a *Personal Access Token* (PAT) for use with automation controller:
    a. In the profile settings of your GitHub account, select **Settings**.
    b. From the navigation panel, select **<> Developer Settings**.
    c. On the **Developer Settings** page, select **Personal access tokens**.
    d. Select **Tokens(classic)**
    e. From the **Personal access tokens** screen, click **Generate a personal access token**.
    f. When prompted, enter your GitHub account password to continue.
    g. In the **Note** field, enter a brief description about what this PAT is used for.
    h. In the **Select scopes** fields, check the boxes next to **repo:status**, **repo_deployment**, and **public_repo**. The automation webhook only needs repository scope access, with the exception of invites. For more information, see Scopes for OAuth apps documentation.
    i. Click **Generate token**.

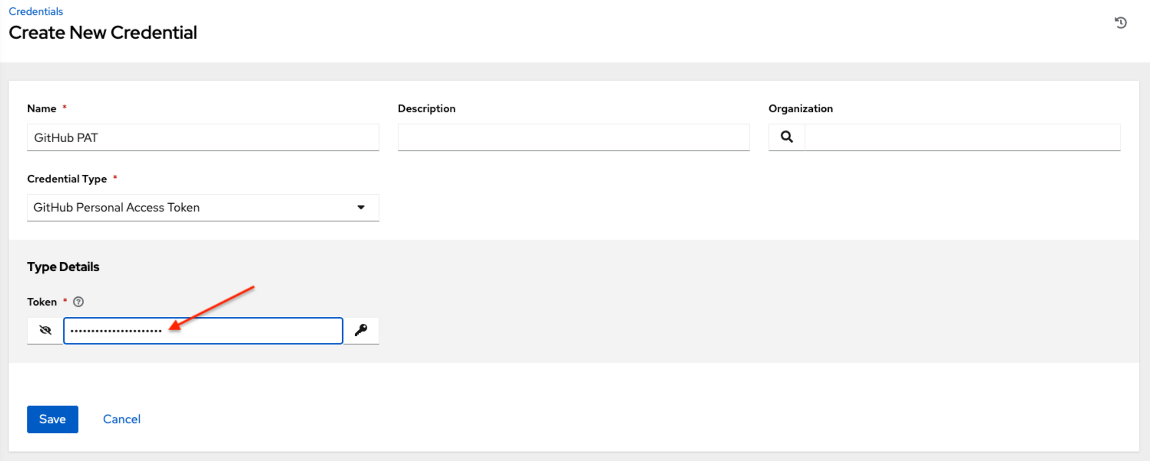
    IMPORTANT

    When the token is generated, ensure that you copy the PAT, as you need it in step 2. You cannot access this token again in GitHub.

2. Use the PAT to optionally create a GitHub credential:

    a. Go to your instance and create a new credential for the GitHub PAT, using the generated token.

    b. Make note of the name of this credential, as you use it in the job template that posts back to GitHub.



Credentials

Create New Credential

Name *: GitHub PAT

Description:

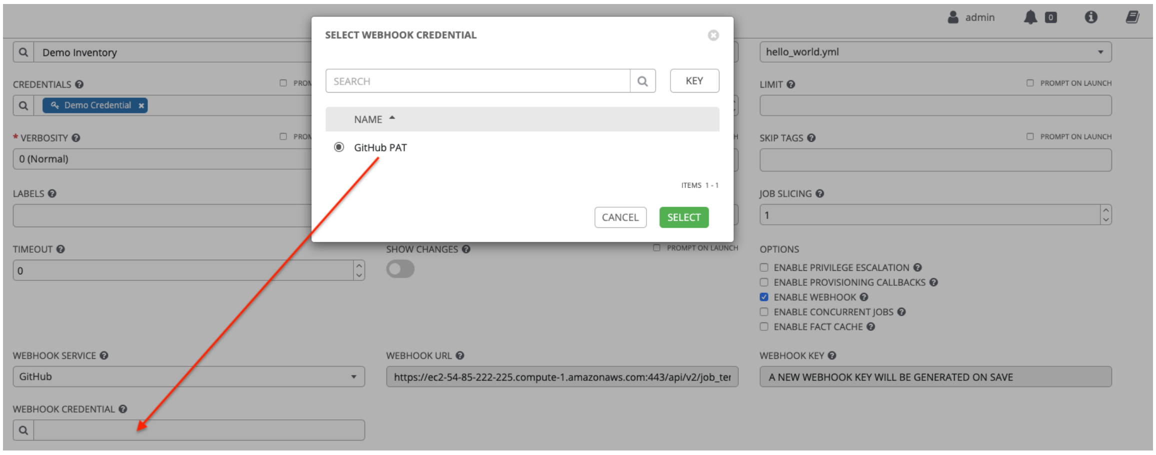
Organization:

Credential Type *: GitHub Personal Access Token

Type Details

Token *: ••••••••••••••••••••

    c. Go to the job template with which you want to enable webhooks, and select the webhook service and credential you created in the preceding step.



SELECT WEBHOOK CREDENTIAL

NAME
● GitHub PAT

ITEMS 1 - 1

Demo Inventory

CREDENTIALS: Demo Credential

* VERBOSITY: 0 (Normal)

LABELS:

TIMEOUT: 0

hello_world.yml

LIMIT: □ PROMPT ON LAUNCH

SKIP TAGS: □ PROMPT ON LAUNCH

JOB SLICING: 1

SHOW CHANGES: □ PROMPT ON LAUNCH

OPTIONS

- □ ENABLE PRIVILEGE ESCALATION
- □ ENABLE PROVISIONING CALLBACKS
- ☑ ENABLE WEBHOOK
- □ ENABLE CONCURRENT JOBS
- □ ENABLE FACT CACHE

WEBHOOK SERVICE: GitHub

WEBHOOK URL: https://ec2-54-85-222-225.compute-1.amazonaws.com:443/api/v2/job_ter

WEBHOOK KEY: A NEW WEBHOOK KEY WILL BE GENERATED ON SAVE

WEBHOOK CREDENTIAL:

    d. Click **Save**. Your job template is set up to post back to GitHub.

3. Go to a GitHub repository where you want to configure webhooks and select **Settings**.

4. From the navigation panel, select **Webhooks → Add webhook**.

5. To complete the **Add webhook** page, you must check the **Enable Webhook** option in a job template or workflow job template. For more information, see step 3 in both Creating a job template and Creating a workflow template.

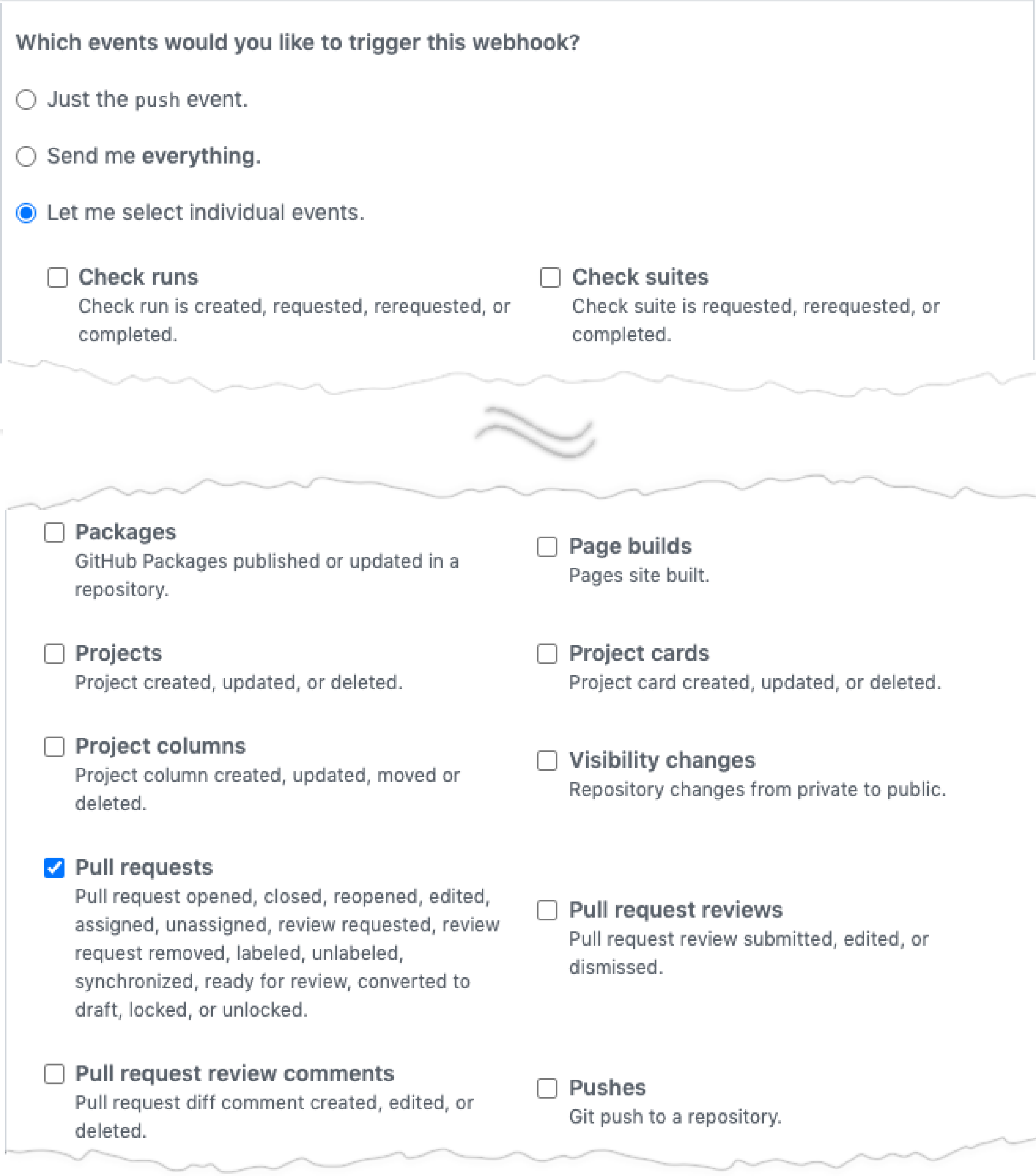
6. Complete the following fields:

    - **Payload URL**: Copy the contents of the **Webhook URL** from the job template and paste it here. The results are sent to this address from GitHub.

    - **Content type**: Set it to **application/json**.

    - **Secret**: Copy the contents of the **Webhook Key** from the job template and paste it here.

    - **Which events would you like to trigger this webhook?** Select the types of events you want to trigger a webhook. Any such event will trigger the job or workflow. To have the job status (pending, error, success) sent back to GitHub, you must select **Pull requests** in the **Let me select individual events** section.



Which events would you like to trigger this webhook?

○ Just the push event.

○ Send me everything.

● Let me select individual events.

□ **Check runs**
Check run is created, requested, rerequested, or completed.

□ **Check suites**
Check suite is requested, rerequested, or completed.

□ **Packages**
GitHub Packages published or updated in a repository.

□ **Page builds**
Pages site built.

□ **Projects**
Project created, updated, or deleted.

□ **Project cards**
Project card created, updated, or deleted.

□ **Project columns**
Project column created, updated, moved or deleted.

□ **Visibility changes**
Repository changes from private to public.

☑ **Pull requests**
Pull request opened, closed, reopened, edited, assigned, unassigned, review requested, review request removed, labeled, unlabeled, synchronized, ready for review, converted to draft, locked, or unlocked.

□ **Pull request reviews**
Pull request review submitted, edited, or dismissed.

□ **Pull request review comments**
Pull request diff comment created, edited, or deleted.

□ **Pushes**
Git push to a repository.

- **Active**: Leave this checked.

7. Click **Add webhook**.

8. When your webhook is configured, it is displayed in the list of webhooks active for your repository, along with the ability to edit or delete it. Click on a webhook, to go to the **Manage webhook** screen.

9. Scroll to view the delivery attempts made to your webhook and whether they succeeded or failed.

Additional resources

For more information, see the Webhooks documentation.

27.2. SETTING UP A GITLAB WEBHOOK

Automation controller has the ability to run jobs based on a triggered webhook event coming in. Job status information (pending, error, success) can be sent back only for pull request events. If automation controller is not required to post job statuses back to the webhook service, go directly to step 3.

Procedure

1. Generate a *Personal Access Token* (PAT) for use with automation controller:

    a. From the navigation panel in GitLab, select your avatar and **Edit profile**.

    b. From the navigation panel, select **Access tokens**.

    c. Complete the following fields:

    - **Token name**: Enter a brief description about what this PAT is used for.
    - **Expiration date**: Skip this field unless you want to set an expiration date for your webhook.
    - **Select scopes**: Select those that are applicable to your integration. For automation controller, **api** is the only selection necessary.

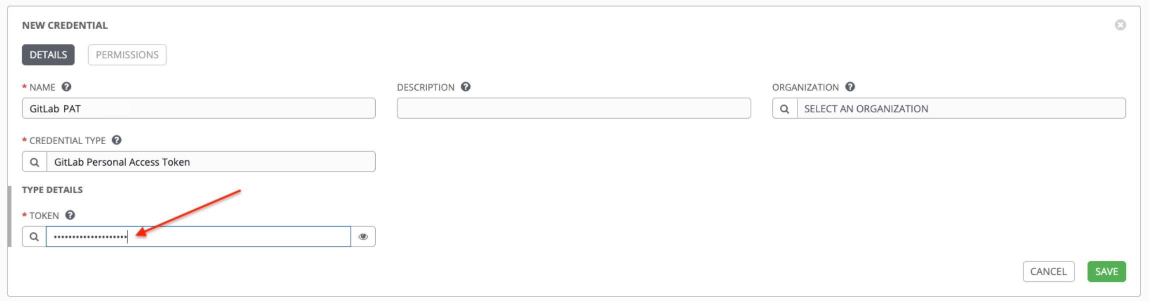
    d. Click **Create personal access token**.

    > **IMPORTANT**
    >
    > When the token is generated, ensure that you copy the PAT, as you need it in step 2. You cannot access this token again in GitLab.

2. Use the PAT to optionally create a GitLab credential:

    a. Go to your instance, and create a new credential for the GitLab PAT, using the generated token.

    b. Make note of the name of this credential, as you use it in the job template that posts back to GitLab.



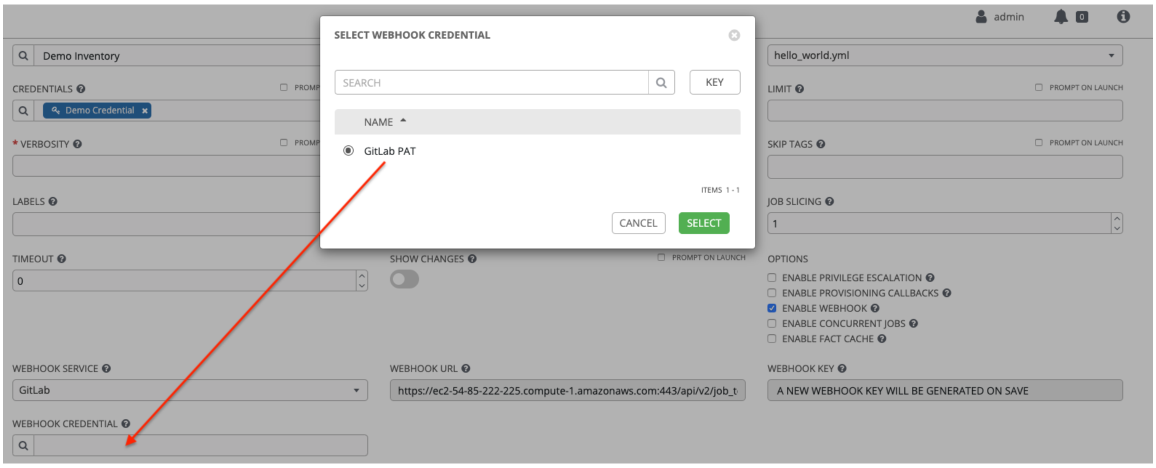
NEW CREDENTIAL

* NAME: GitLab PAT
DESCRIPTION:
ORGANIZATION: SELECT AN ORGANIZATION
* CREDENTIAL TYPE: GitLab Personal Access Token

TYPE DETAILS

* TOKEN: ••••••••••••••••••••

    c. Go to the job template with which you want to enable webhooks, and select the webhook service and credential you created in the preceding step.



SELECT WEBHOOK CREDENTIAL

NAME

● GitLab PAT

ITEMS 1 - 1

Demo Inventory

CREDENTIALS: Demo Credential

hello_world.yml

JOB SLICING: 1

TIMEOUT: 0

OPTIONS

□ ENABLE PRIVILEGE ESCALATION

□ ENABLE PROVISIONING CALLBACKS

☑ ENABLE WEBHOOK

□ ENABLE CONCURRENT JOBS

□ ENABLE FACT CACHE

WEBHOOK SERVICE: GitLab

WEBHOOK URL: https://ec2-54-85-222-225.compute-1.amazonaws.com:443/api/v2/job_t

WEBHOOK KEY: A NEW WEBHOOK KEY WILL BE GENERATED ON SAVE

WEBHOOK CREDENTIAL

d. Click **Save**. Your job template is set up to post back to GitLab.

3. Go to a GitLab repository where you want to configure webhooks.

4. From the navigation panel, select **Settings → Integrations**.

5. To complete the **Add webhook** page, you must check the **Enable Webhook** option in a job template or workflow job template. For more information, see step 3 in both Creating a job template and Creating a workflow template.

6. Complete the following fields:

   - **URL**: Copy the contents of the **Webhook URL** from the job template and paste it here. The results are sent to this address from GitLab.

   - **Secret Token**: Copy the contents of the **Webhook Key** from the job template and paste it here.

   - **Trigger**: Select the types of events you want to trigger a webhook. Any such event will trigger the job or workflow. To have job status (pending, error, success) sent back to GitLab, you must select **Merge request events** in the **Trigger** section.

   - **SSL verification**: Leave **Enable SSL verification** selected.

7. Click **Add webhook**.

8. When your webhook is configured, it is displayed in the list **Project Webhooks** for your repository, along with the ability to test events, edit or delete the webhook. Testing a webhook event displays the results on each page whether it succeeded or failed.

Additional resources

For more information, see Webhooks.

27.3. VIEWING THE PAYLOAD OUTPUT

You can view the entire payload exposed as an extra variable.

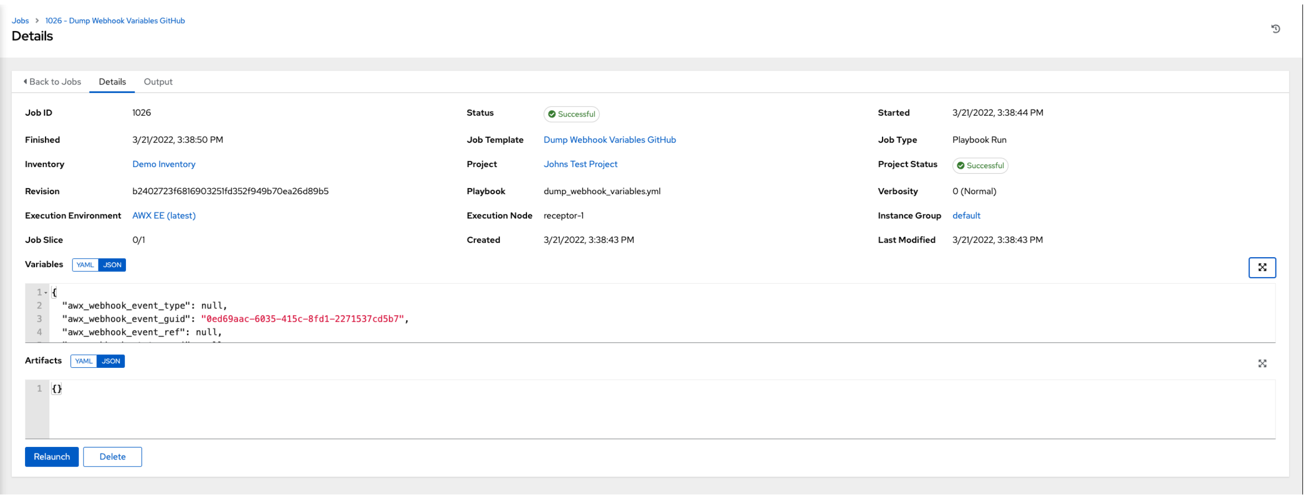
Procedure

1. From the navigation panel, select **Automation Execution → Jobs**.

2. Select the job template with the webhook enabled.

3. Select the **Details** tab.

4. In the **Extra Variables** field, view the payload output from the **awx_webhook_payload** variable, as shown in the following example:



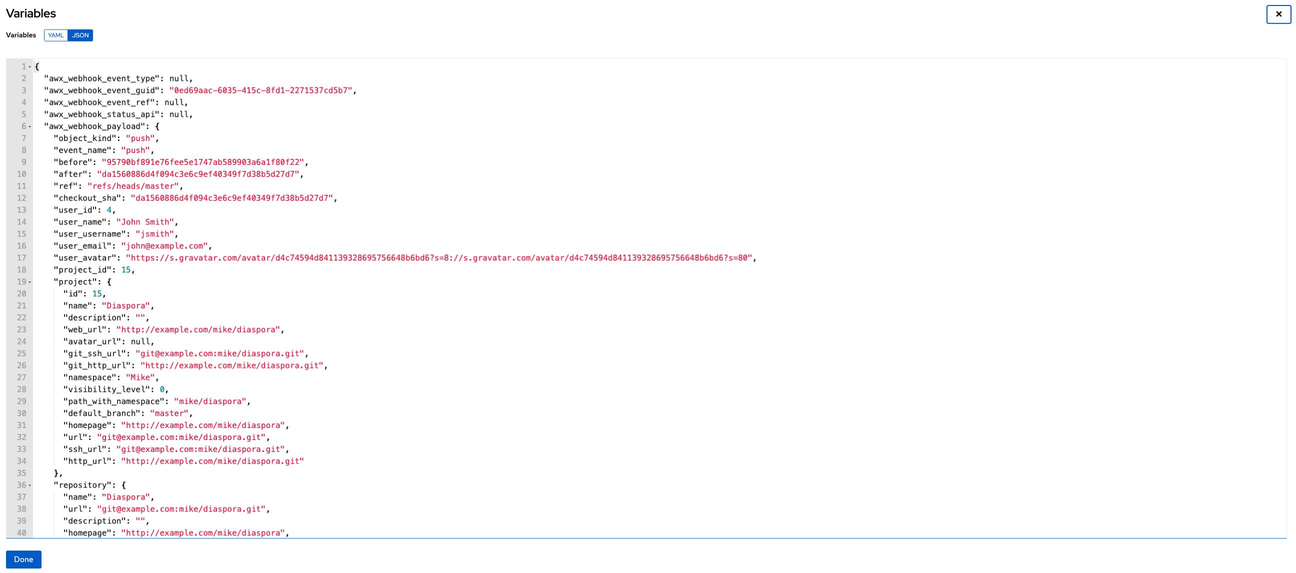
Job ID	1026	**Status**	Successful	**Started**	3/21/2022, 3:38:44 PM
Finished	3/21/2022, 3:38:50 PM	**Job Template**	Dump Webhook Variables GitHub	**Job Type**	Playbook Run
Inventory	Demo Inventory	**Project**	Johns Test Project	**Project Status**	Successful
Revision	b2402723f6816903251fd352f949b70ea26d89b5	**Playbook**	dump_webhook_variables.yml	**Verbosity**	0 (Normal)
Execution Environment	AWX EE (latest)	**Execution Node**	receptor-1	**Instance Group**	default
Job Slice	0/1	**Created**	3/21/2022, 3:38:43 PM	**Last Modified**	3/21/2022, 3:38:43 PM

Variables YAML JSON

```
{
  "awx_webhook_event_type": null,
  "awx_webhook_event_guid": "0ed69aac-6035-415c-8fd1-2271537cd5b7",
  "awx_webhook_event_ref": null,
```

Artifacts YAML JSON

```
{}
```



Variables YAML JSON

```
{
  "awx_webhook_event_type": null,
  "awx_webhook_event_guid": "0ed69aac-6035-415c-8fd1-2271537cd5b7",
  "awx_webhook_event_ref": null,
  "awx_webhook_status_api": null,
  "awx_webhook_payload": {
    "object_kind": "push",
    "event_name": "push",
    "before": "95790bf891e76fee5e1747ab589903a6a1f80f22",
    "after": "da1560886d4f094c3e6c9ef40349f7d38b5d27d7",
    "ref": "refs/heads/master",
    "checkout_sha": "da1560886d4f094c3e6c9ef40349f7d38b5d27d7",
    "user_id": 4,
    "user_name": "John Smith",
    "user_username": "jsmith",
    "user_email": "john@example.com",
    "user_avatar": "https://s.gravatar.com/avatar/d4c74594d841139328695756648b6bd6?s=8://s.gravatar.com/avatar/d4c74594d841139328695756648b6bd6?s=80",
    "project_id": 15,
    "project": {
      "id": 15,
      "name": "Diaspora",
      "description": "",
      "web_url": "http://example.com/mike/diaspora",
      "avatar_url": null,
      "git_ssh_url": "git@example.com:mike/diaspora.git",
      "git_http_url": "http://example.com/mike/diaspora.git",
      "namespace": "Mike",
      "visibility_level": 0,
      "path_with_namespace": "mike/diaspora",
      "default_branch": "master",
      "homepage": "http://example.com/mike/diaspora",
      "url": "git@example.com:mike/diaspora.git",
      "ssh_url": "git@example.com:mike/diaspora.git",
      "http_url": "http://example.com/mike/diaspora.git"
    },
    "repository": {
      "name": "Diaspora",
      "url": "git@example.com:mike/diaspora.git",
      "description": "",
      "homepage": "http://example.com/mike/diaspora",
```

CHAPTER 28. SETTING UP RED HAT INSIGHTS FOR RED HAT ANSIBLE AUTOMATION PLATFORM REMEDIATIONS

Automation controller supports integration with Red Hat Insights.

When a host is registered with Red Hat Insights, it is scanned continually for vulnerabilities and known configuration conflicts. Each problem identified can have an associated fix in the form of an Ansible Playbook.

Red Hat Insights users create a maintenance plan to group the fixes and can create a playbook to mitigate the problems. Automation controller tracks the maintenance plan playbooks through a Red Hat Insights project.

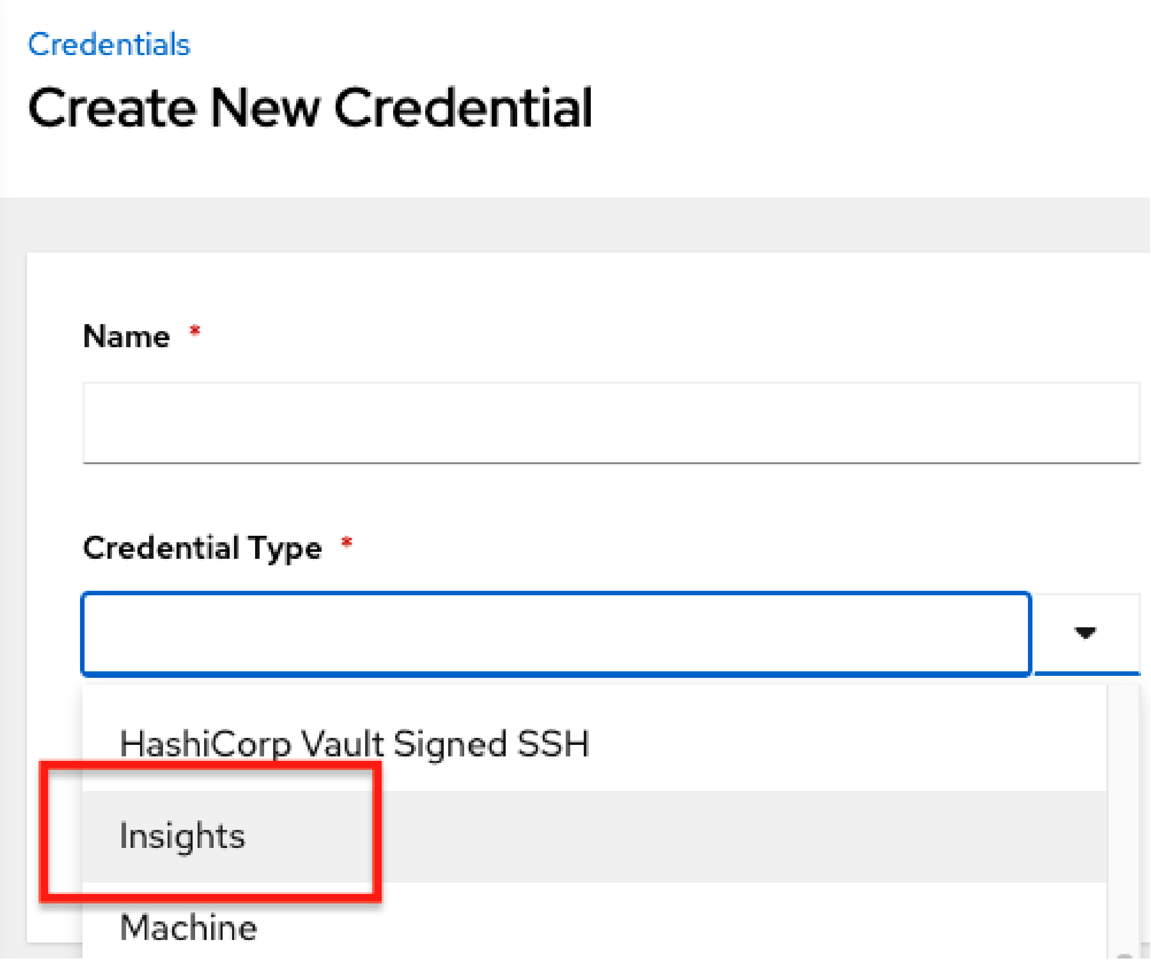
Authentication to Red Hat Insights through Basic Authorization is backed by a special credential, which must first be established in automation controller. To run a Red Hat Insights maintenance plan, you need a Red Hat Insights project and inventory.

28.1. CREATING RED HAT INSIGHTS CREDENTIALS

Use the following procedure to create a new credential for use with Red Hat Insights:

Procedure

1. From the navigation panel, select **Automation Execution → Infrastructure → Credentials**.
2. Click **Create credential**.
3. Enter the appropriate details in the following fields:
   - **Name**: Enter the name of the credential.
   - Optional: **Description**: Enter a description for the credential.
   - Optional: **Organization**: Enter the name of the organization with which the credential is associated, or click the search 🔍 icon and select it from the **Select organization** window.
   - **Credential type**: Enter **Insights** or select it from the list.



Credentials

Create New Credential

Name *

Credential Type *

HashiCorp Vault Signed SSH

Insights

Machine

- **Username**: Enter a valid Red Hat Insights credential.

- **Password**: Enter a valid Red Hat Insights credential. The Red Hat Insights credentials are the user's Red Hat Customer Portal account username and password.

4. Click **Create credential**.

28.2. CREATING A RED HAT INSIGHTS PROJECT

Use the following steps to create a new project for use with Red Hat Insights:

Procedure

1. From the navigation panel, select **Automation Execution → Projects**.

2. Click **Create project**.

3. Enter the appropriate details in the following fields. Note that the following fields require specific Red Hat Insights related entries:

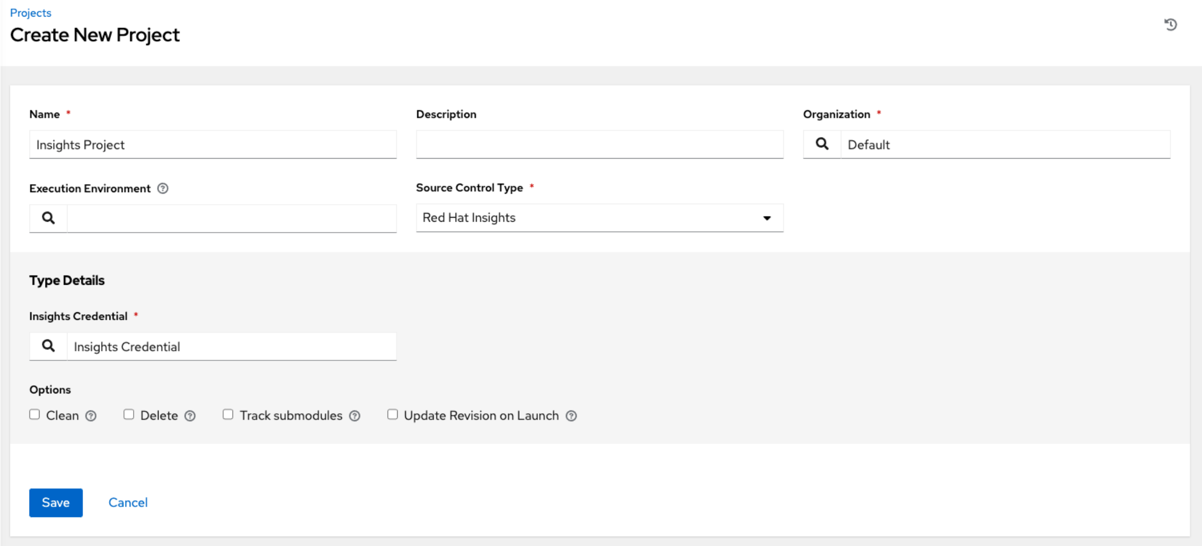
   - **Name**: Enter the name for your Red Hat Insights project.

   - Optional: **Description**: Enter a description for the project.

   - **Organization**: Enter the name of the organization with which the credential is associated, or click the search 🔍 icon and select it from the **Select organization** window.

- Optional: **Execution environment**: The execution environment that is used for jobs that use this project.
- **Source control type**: Select **Red Hat Insights**.
- Optional: **Content signature validation credential**: Enable content signing to verify that the content has remained secure when a project is synced.
- **Insights credential**: This is pre-populated with the Red Hat Insights credential you created before. If not, enter the credential, or click the search 🔍 icon and select it from the **Select Insights Credential** window.

4. Select the update options for this project from the **Options** field and provide any additional values, if applicable. For more information about each option click the tooltip ❔ icon next to each one.



Projects

Create New Project

Name *

Insights Project

Description

Organization *

🔍 Default

Execution Environment ⓘ

🔍

Source Control Type *

Red Hat Insights ▾

Type Details

Insights Credential *

🔍 Insights Credential

Options

☐ Clean ⓘ ☐ Delete ⓘ ☐ Track submodules ⓘ ☐ Update Revision on Launch ⓘ

5. Click **Create project**.

All SCM and project synchronizations occur automatically the first time you save a new project. If you want them to be updated to what is current in Red Hat Insights, manually update the SCM-based project by clicking the update ☁ icon under the project's available actions.

This process syncs your Red Hat Insights project with your Red Hat Insights account solution. Note that the status dot beside the name of the project updates once the sync has run.

28.3. CREATE AN INSIGHTS INVENTORY

The Insights playbook contains a **hosts:** line where the value is the host name supplied to red Hat insights, which can be different from the host name supplied to automation controller

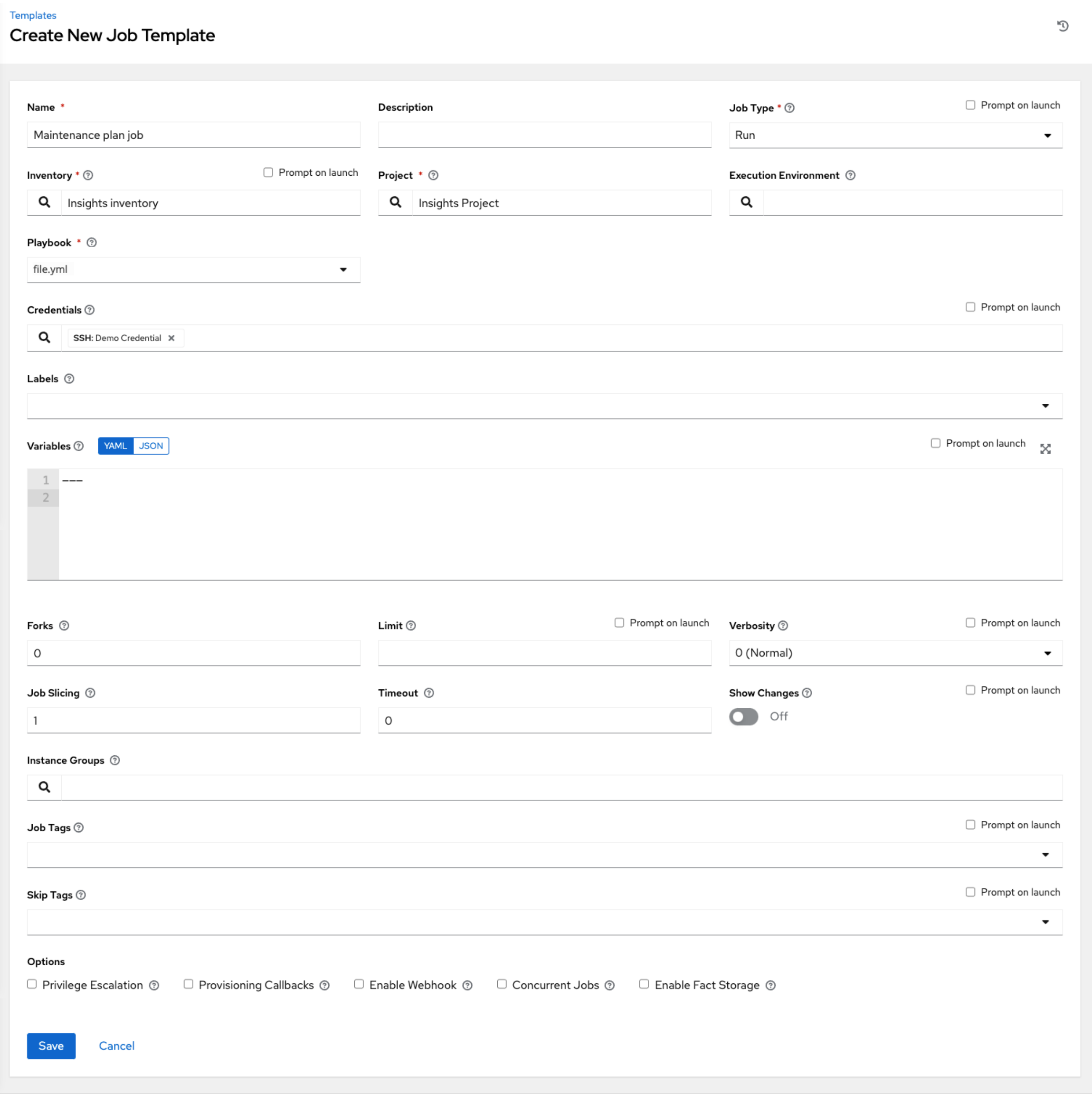
To create a new inventory to use with Red Hat Insights, see Creating Insights credentials.

28.4. REMEDIATING A RED HAT INSIGHTS INVENTORY

Remediation of a Red Hat Insights inventory enables automation controller to run Red Hat Insights playbooks with a single click. You can do this by creating a job template to run the Red Hat Insights remediation.

Procedure

1. From the navigation menu, select **Automation Execution** → **Templates**.
2. On the **Templates** list view, click **Create template** and select from the list.
3. Enter the appropriate details in the following fields. Note that the following fields require specific Red Hat Insights related entries:
    - **Name**: Enter the name of your Maintenance Plan.
    - Optional: **Description**: Enter a description for the job template.
    - **Job Type**: If not already populated, select **Run** from the job type list.
    - **Inventory**: Select the Red Hat Insights inventory that you previously created.
    - **Project**: Select the Red Hat Insights project that you previously created.
    - Optional: **Execution Environment**: The container image to be used for execution.
    - **Playbook**: Select a playbook associated with the Maintenance Plan that you want to run from the playbook list.
    - Optional: **Credentials**: Enter the credential to use for this project or click the search (🔍) icon and select it from the pop-up window. The credential does not have to be a Red Hat Insights credential.
    - **Verbosity**: Keep the default setting, or select the desired verbosity from the list.



Templates

Create New Job Template

Name * : Maintenance plan job

Description :

Job Type * : Run — □ Prompt on launch

Inventory * : Insights inventory — □ Prompt on launch

Project * : Insights Project

Execution Environment :

Playbook * : file.yml

Credentials : SSH: Demo Credential — □ Prompt on launch

Labels :

Variables (YAML | JSON) — □ Prompt on launch

```
---
```

Forks : 0

Limit : — □ Prompt on launch

Verbosity : 0 (Normal) — □ Prompt on launch

Job Slicing : 1

Timeout : 0

Show Changes : Off — □ Prompt on launch

Instance Groups :

Job Tags : — □ Prompt on launch

Skip Tags : — □ Prompt on launch

Options

□ Privilege Escalation □ Provisioning Callbacks □ Enable Webhook □ Concurrent Jobs □ Enable Fact Storage

4. Click **Create job template**.

5. Click the launch 🚀 icon to launch the job template.

When complete, the job results in the **Job Details** page.

CHAPTER 29. BEST PRACTICES FOR AUTOMATION CONTROLLER

The following describes best practice for the use of automation controller:

29.1. USE SOURCE CONTROL

Automation controller supports playbooks stored directly on the server. Therefore, you must store your playbooks, roles, and any associated details in source control. This way you have an audit trail describing when and why you changed the rules that are automating your infrastructure. Additionally, it permits sharing of playbooks with other parts of your infrastructure or team.

29.2. ANSIBLE FILE AND DIRECTORY STRUCTURE

If you are creating a common set of roles to use across projects, these should be accessed through source control submodules, or a common location such as /**opt**. Projects should not expect to import roles or content from other projects.

For more information, see the link General tips from the Ansible documentation.

NOTE

- Avoid using the playbooks **vars_prompt** feature, as automation controller does not interactively permit **vars_prompt** questions. If you cannot avoid using **vars_prompt**, see the Surveys functionality.

- Avoid using the playbooks **pause** feature without a timeout, as automation controller does not permit canceling a pause interactively. If you cannot avoid using **pause**, you must set a timeout.

Jobs use the playbook directory as the current working directory, although jobs must be coded to use the **playbook_dir** variable rather than relying on this.

29.3. USE DYNAMIC INVENTORY SOURCES

If you have an external source of truth for your infrastructure, whether it is a cloud provider or a local CMDB, it is best to define an inventory sync process and use the support for dynamic inventory (including cloud inventory sources). This ensures your inventory is always up to date.

NOTE

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

29.4. VARIABLE MANAGEMENT FOR INVENTORY

Keep variable data with the hosts and groups definitions (see the inventory editor), rather than using **group_vars**/ and **host_vars**/. If you use dynamic inventory sources, automation controller can synchronize such variables with the database as long as the **Overwrite Variables** option is not set.

29.5. AUTOSCALING

Use the "callback" feature to permit newly booting instances to request configuration for auto-scaling scenarios or provisioning integration.

29.6. LARGER HOST COUNTS

Set "forks" on a job template to larger values to increase parallelism of execution runs.

29.7. CONTINUOUS INTEGRATION / CONTINUOUS DEPLOYMENT

For a Continuous Integration system, such as Jenkins, to spawn a job, it must make a **curl** request to a job template. The credentials to the job template must not require prompting for any particular passwords. For configuration and use instructions, see Installation in the Ansible documentation.

CHAPTER 30. GLOSSARY

Ad Hoc

Ad hoc refers to using Ansible to perform a quick command, using /usr/bin/ansible, rather than the orchestration language, which is **/usr/bin/ansible-playbook**. An example of an ad hoc command might be rebooting 50 machines in your infrastructure. Anything you can do ad hoc can be accomplished by writing a Playbook. Playbooks can also glue lots of other operations together.

Callback Plugin

Refers to user-written code that can intercept results from Ansible and act on them. Some examples in the GitHub project perform custom logging, send email, or play sound effects.

Control Groups

Also known as '*cgroups*', a control group is a feature in the Linux kernel that enables resources to be grouped and allocated to run processes. In addition to assigning resources to processes, cgroups can also report use of resources by all processes running inside of the cgroup.

Check Mode

Refers to running Ansible with the **--check** option, which does not make any changes on the remote systems, but only outputs the changes that might occur if the command ran without this flag. This is analogous to so-called "dry run" modes in other systems. However, this does not take into account unexpected command failures or cascade effects (which is true of similar modes in other systems). Use Check mode to get an idea of what might happen, but it is not a substitute for a good staging environment.

Container Groups

Container Groups are a type of Instance Group that specify a configuration for provisioning a pod in a Kubernetes or OpenShift cluster where a job is run. These pods are provisioned on-demand and exist only for the duration of the playbook run.

Credentials

Authentication details that can be used by automation controller to launch jobs against machines, to synchronize with inventory sources, and to import project content from a version control system. For more information, see [Credentials].

Credential Plugin

Python code that contains definitions for an external credential type, its metadata fields, and the code needed for interacting with a secret management system.

Distributed Job

A job that consists of a job template, an inventory, and slice size. When executed, a distributed job slices each inventory into a number of "slice size" chunks, which are then used to run smaller job slices.

External Credential Type

A managed credential type used for authenticating with a secret management system.

Facts

Facts are things that are discovered about remote nodes. While they can be used in playbooks and templates just like variables, facts are things that are inferred, rather than set. Facts are automatically discovered when running plays by executing the internal setup module on the remote nodes. You never have to call the setup module explicitly: it just runs. It can be disabled to save time if it is not required. For the convenience of users who are switching from other configuration management systems, the fact module also pulls in facts from the **ohai** and **facter** tools if they are installed, which are fact libraries from Chef and Puppet, respectively.

Forks

Ansible and automation controller communicate with remote nodes in parallel. The level of parallelism can be set in several ways during the creation or editing of a Job Template, by passing **--forks**, or by editing the default in a configuration file. The default is a very conservative five forks, though if you have a lot of RAM, you can set this to a higher value, such as 50, for increased parallelism.

Group

A set of hosts in Ansible that can be addressed as a set, of which many can exist within a single Inventory.

Group Vars

The **group_vars**/ files are files that are stored in a directory with an inventory file, with an optional filename named after each group. This is a convenient place to put variables that are provided to a given group, especially complex data structures, so that these variables do not have to be embedded in the inventory file or playbook.

Handlers

Handlers are like regular tasks in an Ansible playbook (see **Tasks**), but are only run if the Task contains a "notify" directive and also indicates that it changed something. For example, if a configuration file is changed then the task referencing the configuration file templating operation might notify a service restart handler. This means services can be bounced only if they need to be restarted. Handlers can be used for things other than service restarts, but service restarts are the most common use.

Host

A system managed by automation controller, which may include a physical, virtual, or cloud-based server, or other device (typically an operating system instance). Hosts are contained in an Inventory. Sometimes referred to as a "node".

Host Specifier

Each Play in Ansible maps a series of tasks (which define the role, purpose, or orders of a system) to a set of systems. This "hosts:" directive in each play is often called the hosts specifier. It can select one system, many systems, one or more groups, or hosts that are in one group and explicitly not in another.

Instance Group

A group that contains instances for use in a clustered environment. An instance group provides the ability to group instances based on policy.

Inventory

A collection of hosts against which Jobs can be launched.

Inventory Script

A program that looks up hosts, group membership for hosts, and variable information from an external resource, whether that be a SQL database, a CMDB solution, or LDAP. This concept was adapted from Puppet (where it is called an "External Nodes Classifier") and works in a similar way.

Inventory Source

Information about a cloud or other script to be merged into the current inventory group, resulting in the automatic population of Groups, Hosts, and variables about those groups and hosts.

Job

One of many background tasks launched by automation controller, this is usually the instantiation of a Job Template, such as the launch of an Ansible playbook. Other types of jobs include inventory imports, project synchronizations from source control, or administrative cleanup actions.

Job Detail

The history of running a particular job, including its output and success/failure status.

Job Slice

See **Distributed Job**.

Job Template

The combination of an Ansible playbook and the set of parameters required to launch it. For more information, see Job templates.

JSON

JSON is a text-based format for representing structured data based on JavaScript object syntax. Ansible and automation controller use JSON for return data from remote modules. This enables modules to be written in any language, not just Python.

Mesh

Describes a network comprising of nodes. Communication between nodes is established at the transport layer by protocols such as TCP, UDP or Unix sockets.

See also, **Node**.

Metadata

Information for locating a secret in the external system once authenticated. The user provides this information when linking an external credential to a target credential field.

Node

A node corresponds to entries in the instance database model, or the **/api/v2/instances/** endpoint, and is a machine participating in the cluster or mesh. The unified jobs API reports **controller_node** and **execution_node** fields. The execution node is where the job runs, and the controller node interfaces between the job and server functions.

Node Type	Description
Control	Nodes that run persistent services, and delegate jobs to hybrid and execution nodes.
Hybrid	Nodes that run persistent services and execute jobs.
Hop	Used for relaying across the mesh only.
Execution	Nodes that run jobs delivered from control nodes (jobs submitted from the user’s Ansible automation)

Notification Template

An instance of a notification type (Email, Slack, Webhook, etc.) with a name, description, and a defined configuration.

Notification

A Notification, such as Email, Slack or a Webhook, has a name, description and configuration defined in a Notification template. For example, when a job fails, a notification is sent using the configuration defined by the notification template.

Notify

The act of a task registering a change event and informing a handler task that another action needs to be run at the end of the play. If a handler is notified by multiple tasks, it is still only run once. Handlers are run in the order they are listed, not in the order that they are notified.

Organization

A logical collection of Users, Teams, Projects, and Inventories. Organization is the highest level in the object hierarchy.

Organization Administrator

An user with the rights to modify the Organization’s membership and settings, including making new users and projects within that organization. An organization administrator can also grant permissions to other users within the organization.

Permissions

The set of privileges assigned to Users and Teams that provide the ability to read, modify, and administer Projects, Inventories, and other objects.

Plays

A play is minimally a mapping between a set of hosts selected by a host specifier (usually chosen by groups, but sometimes by hostname globs) and the tasks which run on those hosts to define the role that those systems perform. A playbook is a list of plays. There can be one or many plays in a playbook.

Playbook

An Ansible playbook. For more information, see Ansible playbooks.

Policy

Policies dictate how instance groups behave and how jobs are executed.

Project

A logical collection of Ansible playbooks, represented in automation controller.

Roles

Roles are units of organization in Ansible and automation controller. Assigning a role to a group of hosts (or a set of groups, or host patterns, etc.) implies that they implement a specific behavior. A role can include applying variable values, tasks, and handlers, or a combination of these things. Because of the file structure associated with a role, roles become redistributable units that enable you to share behavior among playbooks, or with other users.

Secret Management System

A server or service for securely storing and controlling access to tokens, passwords, certificates, encryption keys, and other sensitive data.

Schedule

The calendar of dates and times for which a job should run automatically.

Sliced Job

See **Distributed Job**.

Source Credential

An external credential that is linked to the field of a target credential.

Sudo

Ansible does not require root logins and, since it is daemonless, does not require root level daemons (which can be a security concern in sensitive environments). Ansible can log in and perform many operations wrapped in a **sudo** command, and can work with both password-less and password-based sudo. Some operations that do not normally work with **sudo** (such as **scp** file transfer) can be achieved with Ansible’s *copy*, *template*, and *fetch* modules while running in **sudo** mode.

Superuser

An administrator of the server who has permission to edit any object in the system, whether or not it is associated with any organization. Superusers can create organizations and other superusers.

Survey

Questions asked by a job template at job launch time, configurable on the job template.

Target Credential

A non-external credential with an input field that is linked to an external credential.

Team

A sub-division of an Organization with associated Users, Projects, Credentials, and Permissions. Teams provide a means to implement role-based access control schemes and delegate responsibilities across Organizations.

User

An operator with associated permissions and credentials.

Webhook

Webhooks enable communication and information sharing between applications. They are used to respond to commits pushed to SCMs and launch job templates or workflow templates.

Workflow Job Template

A set consisting of any combination of job templates, project syncs, and inventory syncs, linked together in order to execute them as a single unit.

YAML

A human-readable language that is often used for writing configuration files. Ansible and automation controller use YAML to define playbook configuration languages and also variable files. YAML has a minimum of syntax, is very clean, and is easy for people to skim. It is a good data format for configuration files and humans, but is also machine readable. YAML is popular in the dynamic language community and the format has libraries available for serialization in many languages. Examples include Python, Perl, or Ruby.