

Configuring automation execution

This guide describes the administration of automation controller through custom scripts, management jobs, and more. Written for DevOps engineers and administrators, the Configuring automation execution guide assumes a basic understanding of the systems requiring management with automation controllers easy-to-use graphical interface.

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.

1. Start, stop, and restart automation controller

Automation controller includes an administrator utility script, `automation-controller-service`. The script can start, stop, and restart all automation controller services running on the current single automation controller node. The script includes the message queue components and the database if it is an integrated installation.

External databases must be explicitly managed by the administrator. You can find the services script in `/usr/bin/automation-controller-service`, which can be invoked with the following command:

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

NOTE

In clustered installs, the `automation-controller-service restart` does not include PostgreSQL as part of the services that are restarted. This is because it exists external to automation controller, and PostgreSQL does not always need to be restarted. Use `systemctl restart automation-controller` to restart services on clustered environments instead.

You must also restart each cluster node for certain changes to persist as opposed to a single node for a localhost install.

For more information on clustered environments, see the [Clustering](#) section.

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

IMPORTANT

When running automation controller in a container, do not use the `automation-controller-service` script. Restart the pod using the container environment instead.

2. Automation controller configuration

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

Save applies the changes you make, but it does not exit the edit dialog.

To return to the **Settings** page, from the navigation panel select Settings or use the breadcrumbs at the top of the current view.

2.1. Configuring system settings

You can use the **System** menu to define automation controller system settings.

Procedure

1. From the navigation panel, select **Settings** › **Automation Execution** › **System**. The **System Settings** page is displayed.
2. Click **[Edit]**.
3. You can configure the following options:

- **Base URL of the service:** This setting is used by services such as notifications to render a valid URL to the service.
- **Proxy IP allowed list:** If the service is behind a reverse proxy or load balancer, use this setting to configure the proxy IP addresses from which the service should trust custom `REMOTE_HOST_HEADERS` header values.

If this setting is an empty list (the default), the headers specified by `REMOTE_HOST_HEADERS` are trusted unconditionally.

- **Red Hat Client ID for analytics** Client ID used to send data to Automation Analytics.
- **Red Hat Client Secret for analytics** Client secret used to send data to Automation Analytics.
- **Red Hat Client ID for subscriptions** Client ID used to retrieve subscription and content information.
- **Red Hat Client Secret for Subscriptions** Client secret used to retrieve subscription and content information
- **Global default execution environment:** The execution environment to be used when one has not been configured for a job template.
- **Custom virtual environment paths:** Paths where automation controller looks for custom virtual environments.

Enter one path per line.

- **Automation Analytics Gather Interval:** Interval (in seconds) between data gathering.

If **Gather data for Automation Analytics** is set to false, this value is ignored.

- **Remote Host Headers:** HTTP headers and meta keys to search to decide remote hostname or IP. Add additional items to this list, such as `HTTP_X_FORWARDED_FOR`, if behind a reverse proxy. For more information, see [Configuring proxy support for Red Hat Ansible Automation Platform](#).
- **Automation Analytics upload URL:** This value has been set manually in a settings file. This setting is used to configure the upload URL for data collection for Automation Analytics.
- **Defines subscription usage model and shows Host Metrics:**

You can select the following options:

- No subscription: Deletion of host_metrics will not be considered for purposes of managed host counting.
- Usage based on unique managed nodes in a large historical time frame and delete functionality for no longer used managed nodes.

4. You can set the following options:

- **Enable Activity Stream:** Set to enable capturing activity for the activity stream.
- **Enable Activity Stream for Inventory Sync:** Set to enable capturing activity for the activity stream when running inventory sync.
- **All Users Visible to Organization Admins:** Set to control whether any organization administrator can view all users and teams, even those not associated with their organization.
- **Organization Admins Can Manage Users and Teams:** Set to control whether an organization administrator has the privileges to create and manage users and teams.

You might want to disable this ability if you are using an LDAP or SAML integration.

- **Gather data for Automation Analytics:** Set to enable the service to gather data on automation and send it to Automation Analytics.

5. Click **[Save]**

2.2. Configuring jobs

You can use the **Job** option to define the operation of Jobs in automation controller.

Procedure

1. From the navigation panel, select **Settings › Automation Execution › Job**.
2. On the **Job Settings** page, click **[Edit]**.

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

3. You can configure the following options:

- **Ansible Modules Allowed For Ad Hoc Jobs:** List of modules allowed to be used by ad hoc jobs.

The directory in which the service creates new temporary directories for job execution and isolation (such as credential files).

- **When can extra variables contain Jinja templates?:** Ansible allows variable substitution through the Jinja2 templating language for `--extra-vars`.

This poses a potential security risk where users with the ability to specify extra vars at job launch time can use Jinja2 templates to run arbitrary Python.

Set this value to either `template` or `never`.

- **Paths to expose to isolated jobs:** List of paths that would otherwise be hidden to expose to isolated jobs.

Enter one path per line. If a path to a specific file is entered, then the entire directory containing that file will be mounted inside the execution environment.

Volumes are mounted from the execution node to the container.

The supported format is `HOST-DIR[:CONTAINER-DIR[:OPTIONS]]`.

- **Extra Environment Variables:** Additional environment variables set for playbook runs, inventory updates, project updates, and notification sending.
- **K8S Ansible Runner Keep-Alive Message Interval:** Only applies to jobs running in a

Container Group.

If not 0, send a message every specified number of seconds to keep the connection open.

- **Environment Variables for Galaxy Commands:** Additional environment variables set for invocations of ansible-galaxy within project updates. Useful if you must use a proxy server for ansible-galaxy but not git.
- **Standard Output Maximum Display Size:** Maximum Size of Standard Output in bytes to display before requiring the output be downloaded.
- **Job Event Standard Output Maximum Display Size:** Maximum Size of Standard Output in bytes to display for a single job or ad hoc command event. stdout ends with ... when truncated.
- **Job Event Maximum Websocket Messages Per Second:** The maximum number of messages to update the UI live job output with per second.

A value of 0 means no limit.

- **Maximum Scheduled Jobs:** Maximum number of the same job template that can be waiting to run when launching from a schedule before no more are created.
- **Ansible Callback Plugins:** List of paths to search for extra callback plugins to be used when running jobs.
- **Default Job Timeout:** If no output is detected from ansible in this number of seconds the execution will be terminated.

Use a value of 0 to indicate that no idle timeout should be imposed.

Enter one path per line.

- **Default Job Idle Timeout:** If no output is detected from ansible in this number of seconds the execution will be terminated.

Use a value of 0 to indicate that no idle timeout should be imposed.

- **Default Inventory Update Timeout:** Maximum time in seconds to allow inventory updates to run.

Use a value of 0 to indicate that no timeout should be imposed.

A timeout set on an individual inventory source will override this.

- **Default Project Update Timeout:** Maximum time in seconds to allow project updates to run.

Use a value of 0 to indicate that no timeout should be imposed.

A timeout set on an individual project will override this.

- **Per-Host Ansible Fact Cache Timeout:** Maximum time, in seconds, that stored Ansible facts are considered valid since the last time they were modified.

Only valid, non-stale, facts are accessible by a playbook.

This does not influence the deletion of `ansible_facts` from the database.

Use a value of 0 to indicate that no timeout should be imposed.

- **Maximum number of forks per job:** Saving a Job Template with more than this number of forks results in an error.

When set to 0, no limit is applied.

- **Job execution path:** Only available in operator-based installations.
- **Container Run Options:** Only available in operator-based installations.

List of options to pass to Podman run example: `['--network', 'slirp4netns:enable_ipv6=true', '--log-level', 'debug']`.

You can set the following options:

- **Run Project Updates With Higher Verbosity:** Select to add the CLI `-vvv` flag to playbook runs of `project_update.yml` used for project updates
- **Enable Role Download:** Select to allow roles to be dynamically downloaded from a `requirements.yml` file for SCM projects.
- **Enable Collection(s) Download:** Select to allow collections to be dynamically downloaded from a `requirements.yml` file for SCM projects.
- **Follow symlinks:** Select to follow symbolic links when scanning for playbooks.

Be aware that setting this to `True` can lead to infinite recursion if a link points to a parent directory of itself.

- **Expose host paths for Container Groups:** Select to expose paths through `hostPath` for the Pods created by a Container Group.

`HostPath` volumes present many security risks, and it is best practice to avoid the use of `HostPaths` when possible.

Ignore Ansible Galaxy SSL Certificate Verification: If set to `true`, certificate validation is not done when installing content from any Galaxy server.

Click the tooltip  icon next to the field that you need additional information about.

For more information about configuring Galaxy settings, see the [Ansible Galaxy Support](#) section of *Using automation execution*.

NOTE	The values for all timeouts are in seconds.
-------------	---

4. Click **[Save]** to apply the settings.

2.3. Logging and aggregation settings

For information about these settings, see [Setting up logging](#).

2.4. Configuring Automation Analytics

When you imported your license for the first time, you were automatically opted in for the collection of data that powers Automation Analytics, a cloud service that is part of the Ansible Automation Platform subscription.

Prerequisites

- A service account created with the **Automation Analytics Viewer** role in console.redhat.com. For more information, see [Creating a service account](#).

Procedure

1. From the navigation panel, select **Settings › Automation Execution › System**.
2. Click **[Edit]**.
3. In the field labeled **Red Hat Client ID for Analytics**, enter the client ID you received when you created your service account to retrieve subscription and content information.
4. In the field labeled **Red Hat Client Secret for Analytics**, enter the client secret you received when you created your service account to send data to Automation Analytics.
5. In the **Options** list select the checkbox to **Gather data for Automation Analytics**.
6. Click **[Save]**.

Verification

After configuring the service account, run a test job to ensure everything is set up correctly.

1. From the navigation panel, select **Automation Execution › Jobs** to launch a job.
2. Check [analytics at console.redhat.com](#) to confirm that the data is being posted.

2.5. Additional settings for automation controller

There are additional advanced settings that can affect automation controller behavior that are not available in the automation controller UI.

For traditional virtual machine based deployments, these settings can be provided to automation controller by creating a file in `/etc/tower/conf.d/custom.py`. When settings are provided to automation controller through file-based settings, the settings file must be present on all control plane nodes. These include all of the hybrid or control type nodes in the `automationcontroller` group in the installer inventory.

For these settings to be effective, restart the service with `automation-controller-service` restart on each node with the settings file. If the settings provided in this file are also visible in the automation controller UI, then they are marked as "Read only" in the UI.

For container-based installations, use `controller_extra_settings` in the [Automation controller variables](#). The containerized version does not support `custom.py`.

2.6. Troubleshooting options

You can use the **Troubleshooting** page to enable or disable certain flags that aid in debugging issues within Ansible Automation Platform.

Procedure

1. From the navigation panel, select **Settings › Automation Execution › Troubleshooting**.
2. The **Troubleshooting** page is displayed.
3. Click **[Edit]**.
4. You can select the following options:
 - **Enable or Disable tmp dir cleanup**: Select this to enable or disable the cleanup of tmp directories generated during execution of a job after job execution completes.
 - **Debug Web Requests**: Select this to enable or disable web request profiling for debugging slow web requests.
 - **Release Receptor Work**: Select this to turn on or off the deletion of job pods after they complete or fail. This can be helpful in debugging why a job failed.
 - **Keep receptor work on error**: Select this to prevent receptor work from being released when an error is detected.
5. Click **[Save]** to save your changes.

3. Performance tuning for automation controller

Tune your automation controller to optimize performance and scalability. When planning your workload, ensure that you identify your performance and scaling needs, adjust for any limitations, and monitor your deployment.

Automation controller is a distributed system with many components that you can tune, including the following:

- Task system in charge of scheduling jobs
- Control Plane in charge of controlling jobs and processing output
- Execution plane where jobs run
- Web server in charge of serving the API
- WebSocket system that serve and broadcast WebSocket connections and data
- Database used by many components

3.1. Websocket configuration for automation controller

You can configure automation controller to align the websocket configuration with your nginx or load balancer configuration.

Automation controller nodes are interconnected through websockets to distribute all websocket-emitted messages throughout your system. This configuration setup enables any browser client websocket to subscribe to any job that might be running on any automation controller node. Websocket clients are not routed to specific automation controller nodes. Instead, any automation controller node can handle any websocket request and each automation controller node must know about all websocket messages destined for all clients.

You can configure websockets at `/etc/tower/conf.d/websocket_config.py` in all of your automation controller nodes and the changes become effective after the service restarts.

Automation controller automatically handles discovery of other automation controller nodes through the Instance record in the database.

IMPORTANT

Your automation controller nodes are designed to broadcast websocket traffic across a private, trusted subnet (and not the open Internet). Therefore, if you turn off HTTPS for websocket broadcasting, the websocket traffic, composed mostly of Ansible Playbook stdout, is sent unencrypted between automation controller nodes.

3.1.1. Configuring automatic discovery of other automation controller nodes

You can configure websocket connections to enable automation controller to automatically handle discovery of other automation controller nodes through the Instance record in the database.

1. Edit automation controller websocket information for port and protocol, and confirm whether to verify certificates with `True` or `False` when establishing the websocket connections:

```
BROADCAST_WEBSOCKET_PROTOCOL = 'http'
BROADCAST_WEBSOCKET_PORT = 80
BROADCAST_WEBSOCKET_VERIFY_CERT = False
```

2. Restart automation controller with the following command:

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

3.2. Capacity planning for deploying automation controller

Capacity planning for automation controller is planning the scale and characteristics of your deployment so that it has the capacity to run the planned workload. Capacity planning includes the following phases:

1. Characterizing your workload
2. Reviewing the capabilities of different node types
3. Planning the deployment based on the requirements of your workload

3.2.1. Characteristics of your workload

Before planning your deployment, establish the workload that you want to support. Consider the following factors to characterize an automation controller workload:

- Managed hosts
- Tasks per hour per host
- Maximum number of concurrent jobs that you want to support
- Maximum number of forks set on jobs. Forks determine the number of hosts that a job acts on concurrently.
- Maximum API requests per second
- Node size that you prefer to deploy (CPU/Memory/Disk)

3.2.2. Types of nodes in automation controller

You can configure four types of nodes in an automation controller deployment:

- Control nodes
- Hybrid nodes
- Execution nodes
- Hop nodes

However, for an operator-based environment, there are no hybrid or control nodes. There are container groups, which make up containers running on the Kubernetes cluster. That comprises the control plane. That control plane is local to the Kubernetes cluster in which Red Hat Ansible Automation Platform is deployed.

Benefits of scaling control nodes

Control and hybrid nodes provide control capacity. They provide the ability to start jobs and process their output into the database. Every job is assigned a control node. In the default configuration, each job requires one capacity unit to control. For example, a control node with 100 capacity units can control a maximum of 100 jobs.

Vertically scaling a control node by deploying a larger virtual machine with more resources increases the following capabilities of the control plane:

- The number of jobs that a control node can perform control tasks for, which requires both more CPU and memory.
- The number of job events a control node can process concurrently.

Scaling CPU and memory in the same proportion is recommended, for example, 1 CPU: 4 GB RAM. Even when memory consumption is high, increasing the CPU of an instance can often relieve pressure. The majority of the memory that control nodes consume is from unprocessed events that are stored in a memory-based queue.

NOTE

Vertically scaling a control node does not automatically increase the number of workers that handle web requests.

An alternative to vertically scaling is horizontally scaling by deploying more control nodes. This allows spreading control tasks across more nodes and allowing web traffic to be spread over more nodes, given that you provision a load balancer to spread requests across nodes. Horizontally scaling by deploying more control nodes in many ways can be preferable as it additionally provides for more redundancy and workload isolation when a control node goes down or experiences higher than normal load.

Benefits of scaling execution nodes

Execution and hybrid nodes provide execution capacity. The capacity consumed by a job is equal to the number of forks set on the job template or the number of hosts in the inventory, whichever is less, plus one additional capacity unit to account for the main ansible process. For example, a job template with the default forks value of 5 acting on an inventory with 50 hosts consumes 6 capacity units from the execution node it is assigned to.

Vertically scaling an execution node by deploying a larger virtual machine with more resources provides more forks for job execution. This increases the number of concurrent jobs that an instance can run.

In general, scaling CPU alongside memory in the same proportion is recommended. Like control and hybrid nodes, there is a capacity adjustment on each execution node that you can use to align actual use with the estimation of capacity consumption that the automation controller makes. By default, all nodes are set to the top of that range. If actual monitoring data reveals the node to be over-used, decreasing the capacity adjustment can help bring this in line with actual usage.

An alternative to vertically scaling execution nodes is horizontally scaling the execution plane by deploying more virtual machines to be execution nodes. Because horizontally scaling can provide additional isolation of workloads, you can assign different instances to different instance groups. You can then assign these instance groups to organizations, inventories, or job templates. For example, you can configure an instance group that can only be used for running jobs against a certain Inventory. In this scenario, by horizontally scaling the execution plane, you can ensure that lower-priority jobs do not block higher-priority jobs

Benefits of scaling hop nodes

Because hop nodes use very low memory and CPU, vertically scaling these nodes does not impact capacity. Monitor the network bandwidth of any hop node that serves as the sole connection between many execution nodes and the control plane. If bandwidth use is saturated, consider changing the network.

Horizontally scaling by adding more hop nodes could provide redundancy when one hop node goes down, which can allow traffic to continue to flow between the control plane and the execution nodes.

Ratio of control to execution capacity

Assuming default configuration, the maximum recommended ratio of control capacity to execution capacity is 1:5 in traditional VM deployments. This ensures that there is enough control capacity to run jobs on all the execution capacity available and process the output. Any less control capacity in relation to the execution capacity, and it would not be able to launch enough jobs to use the execution capacity.

There are cases in which you might want to modify this ratio closer to 1:1. For example, in cases where a job produces a high level of job events, reducing the amount of execution capacity in relation to the control capacity helps relieve pressure on the control nodes to process that output.

3.3. Example capacity planning exercise

After you have determined the workload capacity that you want to support, you must plan your deployment based on the requirements of the workload. To help you with your deployment, review the following planning exercise.

For this example, the cluster must support the following capacity:

- 300 managed hosts
- 1,000 tasks per hour per host or 16 tasks per minute per host
- 10 concurrent jobs
- Forks set to 5 on playbooks. This is the default.
- Average event size is 1 Mb

The virtual machines have 4 CPU and 16 GB RAM, and disks that have 3000 IOPS.

3.3.1. Example workload requirements

For this example capacity planning exercise, use the following workload requirements:

Execution capacity

- To run the 10 concurrent jobs requires at least 60 units of execution capacity.
 - You calculate this by using the following equation: $(10 \text{ jobs} * 5 \text{ forks}) + (10 \text{ jobs} * 1 \text{ base task impact of a job}) = 60 \text{ execution capacity}$

Control capacity

- To control 10 concurrent jobs requires at least 10 units of control capacity.
- To calculate the number of events per hour that you need to support 300 managed hosts and 1,000 tasks per hour per host, use the following equation:
 - $1000 \text{ tasks} * 300 \text{ managed hosts per hour} = 300,000 \text{ events per hour at minimum.}$
 - You must run the job to see exactly how many events it produces, because this is dependent on the specific task and verbosity. For example, a debug task printing “Hello World” produces 6 job events with the verbosity of 1 on one host. With a verbosity of 3, it produces 34 job events on one host. Therefore, you must estimate that the task produces at least 6 events. This would produce closer to 3,000,000 events per hour, or approximately 833 events per second.

Determining quantity of execution and control nodes needed

To determine how many execution and control nodes you need, reference the experimental results in the following table that shows the observed event processing rate of a single control node with 5 execution nodes of equal size (API Capacity column). The default “forks” setting of job templates is 5, so using this default, the maximum number of jobs a control node can dispatch to execution nodes makes 5 execution nodes of equal CPU/RAM use 100% of their capacity, arriving to the previously mentioned 1:5 ratio of control to execution capacity.

Node	API capacity	Default execution capacity	Default control capacity	Mean event processing rate at 100% capacity usage	Mean events processing rate at 50% capacity usage	Mean event processing rate at 40% capacity usage
4 CPU at 2.5Ghz, 16 GB RAM control node, a maximum of 3000 IOPS disk	about 10 requests per second	n/a	137 jobs	1100 per second	1400 per second	1630 per second
4 CPU at 2.5Ghz, 16 GB RAM execution node, a maximum of 3000 IOPS disk	n/a	137	n/a	n/a	n/a	n/a
4 CPU at 2.5Ghz, 16 GB RAM database node, a maximum of 3000 IOPS disk	n/a	n/a	n/a	n/a	n/a	n/a

Because controlling jobs competes with job event processing on the control node, over-provisioning control capacity can reduce processing times. When processing times are high, you can experience a delay between when the job runs and when you can view the output in the API or UI.

For this example, for a workload on 300 managed hosts, executing 1000 tasks per hour per host, 10 concurrent jobs with forks set to 5 on playbooks, and an average event size 1 Mb, use the following procedure:

- Deploy 1 execution node, 1 control node, 1 database node of 4 CPU at 2.5Ghz, 16 GB RAM, and disks that have about 3000 IOPS.
- Keep the default fork setting of 5 on job templates.
- Use the capacity change feature in the instance view of the UI on the control node to reduce the capacity down to 16, the lowest value, to reserve more of the control node's capacity for processing events.

For more information about workloads with high levels of API interaction, see [Scaling Automation Controller for API Driven Workloads](#). For more information about managing capacity with instances, see [Managing capacity with Instances](#). For more information about operator-based deployments, see [Red Hat Ansible Automation Platform considerations for operator environments](#).

3.4. Performance troubleshooting for automation controller

Users experience many request timeouts (504 or 503 errors), or in general high API latency. In the UI, clients face slow login and long wait times for pages to load. What system is the likely culprit?

- If these issues occur only on login, and you use external authentication, the problem is likely with the integration of your external authentication provider and you should seek Red Hat support.
- For other issues with timeouts or high API latency, see [Web server tuning](#).

Long wait times for job output to load.

- Job output streams from the execution node where the ansible-playbook is actually run to the associated control node. Then the callback receiver serializes this data and writes it to the database. Relevant settings to observe and tune can be found in [Settings for managing job event processing](#) and [PostgreSQL database configuration and maintenance for automation controller](#).
- In general, to resolve this symptom it is important to observe the CPU and memory use of the control nodes. If CPU or memory use is very high, you can either horizontally scale the control plane by deploying more virtual machines to be control nodes that naturally spreads out work more, or to modify the number of jobs a control node will manage at a time. For more information, see [Capacity settings for control and execution nodes](#) for more information.
- Job output delay can occur on initial job runs that use execution environments that have not been pulled into the platform. The output becomes visible after the job run completes.

What can you do to increase the number of jobs that automation controller can run concurrently?

- Factors that cause jobs to remain in “pending” state are:

- **Waiting for “dependencies” to finish:** this includes project updates and inventory updates when “update on launch” behavior is enabled.
- **The “allow_simultaneous” setting of the job template:** if multiple jobs of the same job template are in “pending” status, check the “allow_simultaneous” setting of the job template (“Concurrent Jobs” checkbox in the UI). If this is not enabled, only one job from a job template can run at a time.
- **The “forks” value of your job template:** the default value is 5. The amount of capacity required to run the job is roughly the forks value (some small overhead is accounted for). If the forks value is set to a very large number, this will limit what nodes will be able to run it.
- **Lack of either control or execution capacity:** see “awx_instance_remaining_capacity” metric from the application metrics available on /api/v2/metrics. See [Metrics for monitoring automation controller application](#) for more information about how to check metrics. See [Capacity planning for deploying automation controller](#) for information about how to plan your deployment to handle the number of jobs you are interested in.

Jobs run more slowly on automation controller than on a local machine.

- Some additional overhead is expected, because automation controller might be dispatching your job to a separate node. In this case, automation controller is starting a container and running ansible-playbook there, serializing all output and writing it to a database.
- Project update on launch and inventory update on launch behavior can cause additional delays at job start time.
- Size of projects can impact how long it takes to start the job, as the project is updated on the control node and transferred to the execution node. Internal cluster routing can impact network performance. For more information, see [Internal cluster routing](#).
- Container pull settings can impact job start time. The execution environment is a container that is used to run jobs within it. Container pull settings can be set to “Always”, “Never” or “If not present”. If the container is always pulled, this can cause delays.
- Ensure that all cluster nodes, including execution, control, and the database, have been deployed in instances with storage rated to the minimum required IOPS, because the manner in which automation controller runs ansible and caches event data implicates significant disk I/O. For more information, see [System requirements](#).

Database storage does not stop growing.

- Automation controller has a management job titled “Cleanup Job Details”. By default, it is set to keep 120 days of data and to run once a week. To reduce the amount of data in the database, you can shorten the retention time. For more information, see [Removing old activity stream data](#).
- Running the cleanup job deletes the data in the database. However, the database must at some point perform its vacuuming operation which reclaims storage. See [PostgreSQL database configuration and maintenance for automation controller](#) for more information about database vacuuming.

3.5. Metrics to monitor automation controller

Monitor your automation controller hosts at the system and application levels.

System level monitoring includes the following information:

- Disk I/O
- RAM use
- CPU use
- Network traffic

Application level metrics provide data that the application knows about the system. This data includes the following information:

- How many jobs are running in a given instance
- Capacity information about instances in the cluster
- How many inventories are present
- How many hosts are in those inventories

Using system and application metrics can help you identify what was happening in the application when a service degradation occurred. Information about automation controller's performance over time helps when diagnosing problems or doing capacity planning for future growth.

3.5.1. Metrics for monitoring automation controller application

For application level monitoring, automation controller provides Prometheus-style metrics on an API endpoint `/api/v2/metrics`. Use these metrics to monitor aggregate data about job status and subsystem performance, such as for job output processing or job scheduling.

The metrics endpoint includes descriptions of each metric. Metrics of particular interest for performance include:

- `awx_status_total`
 - Current total of jobs in each status. Helps correlate other events to activity in system.
 - Can monitor upticks in errored or failed jobs.
- `awx_instance_remaining_capacity`
 - Amount of capacity remaining for running additional jobs.
- `callback_receiver_event_processing_avg_seconds`
 - colloquially called “job events lag”.
 - Running average of the lag time between when a task occurred in ansible and when the user is able to see it. This indicates how far behind the callback receiver is in processing events. When this number is very high, users can consider scaling up the control plane or using the capacity adjustment feature to reduce the number of jobs a control node controls.
- `callback_receiver_events_insert_db`

- Counter of events that have been inserted by a node. Can be used to calculate the job event insertion rate over a given time period.
- `callback_receiver_events_queue_size_redis`
 - Indicator of how far behind callback receiver is in processing events. If too high, Redis can cause the control node to run out of memory (OOM).

3.5.2. System level monitoring

Monitoring the CPU and memory use of your cluster hosts is important because capacity management for instances does not introspect into the actual resource usage of hosts. The resource impact of automation jobs depends on what the playbooks are doing. For example, many cloud or networking modules do most of the processing on the execution node, which runs the Ansible Playbook. The impact on the automation controller is very different than if you were running a native module like “yum” where the work is performed on the target hosts where the execution node spends much of the time during this task waiting on results.

If CPU or memory usage is very high, consider lowering the capacity adjustment (available on the instance detail page) on affected instances in the automation controller. This limits how many jobs are run on or controlled by this instance.

Monitor the disk I/O and use of your system. The manner in which an automation controller node runs Ansible and caches output on the file system, and eventually saves it in the database, creates high levels of disk reads and writes. Identifying poor disk performance early can help prevent poor user experience and system degradation.

Additional resources

- [Metrics](#)
- [Automation analytics and Red Hat Insights for Red Hat Ansible Automation Platform](#)

3.6. PostgreSQL database configuration and maintenance for automation controller

To improve the performance of automation controller, you can configure the following configuration parameters in the database:

Maintenance

The `VACUUM` and `ANALYZE` tasks are important maintenance activities that can impact performance. In normal PostgreSQL operation, tuples that are deleted or obsoleted by an update are not physically removed from their table; they remain present until a `VACUUM` is done. Therefore it's necessary to do `VACUUM` periodically, especially on frequently-updated tables. `ANALYZE` collects statistics about the contents of tables in the database, and stores the results in the `pg_statistic` system catalog. Subsequently, the query planner uses these statistics to help determine the most efficient execution plans for queries. The autovacuuming PostgreSQL configuration parameter automates the execution of `VACUUM` and `ANALYZE` commands. Setting autovacuuming to `true` is a good practice. However, autovacuuming will not occur if there is never any idle time on the database. If it is observed that autovacuuming is not sufficiently cleaning up space on the database disk, then

scheduling specific vacuum tasks during specific maintenance windows can be a solution.

Configuration parameters

To improve the performance of the PostgreSQL server, configure the following *Grand Unified Configuration* (GUC) parameters that manage database memory. You can find these parameters inside the `$PGDATA` directory in the `postgresql.conf` file, which manages the configurations of the database server.

- **shared_buffers**: determines how much memory is dedicated to the server for caching data. The default value for this parameter is 128 MB. When you modify this value, you must set it between 15% and 25% of the machine's total RAM.

NOTE If you are compiling Postgres against OpenSSL 3.2, your system regresses to remove the parameter for User during startup. You can rectify this by using the `BIO_get_app_data` call instead of `open_get_data`. Only an administrator can make these changes, but it impacts all users connected to the PostgreSQL database. If you update your systems without the OpenSSL patch, you are not impacted, and you do not need to take action.

NOTE You must restart the database server after changing the value for **shared_buffers**.

WARNING If you are compiling Postgres against OpenSSL 3.2, your system regresses to remove the parameter for User during startup. You can rectify this by using the `BIO_get_app_data` call instead of `open_get_data`. Only an administrator can make these changes, but it impacts all users connected to the PostgreSQL database.

If you update your systems without the OpenSSL patch, you are not impacted, and you do not need to take action.

- **work_mem**: provides the amount of memory to be used by internal sort operations and hash tables before disk-swapping. Sort operations are used for order by, distinct, and merge join operations. Hash tables are used in hash joins and hash-based aggregation. The default value for this parameter is 4 MB. Setting the correct value of the **work_mem** parameter improves the speed of a search by reducing disk-swapping.
 - Use the following formula to calculate the optimal value of the **work_mem** parameter for the database server:

Total RAM * 0.25 / max_connections

NOTE Setting a large **work_mem** can cause the PostgreSQL server to go out of memory (OOM) if there are too many open connections to the database.

- **max_connections**: specifies the maximum number of concurrent connections to the database server.

- `maintenance_work_mem`: provides the maximum amount of memory to be used by maintenance operations, such as vacuum, create index, and alter table add foreign key operations. The default value for this parameter is 64 MB. Use the following equation to calculate a value for this parameter:

Total RAM * 0.05

NOTE Set `maintenance_work_mem` higher than `work_mem` to improve performance for vacuuming.

Additional resources

- [Automatic Vacuuming](#)

3.6.1. Encrypting plain text passwords in automation controller configuration files

Passwords stored in automation controller configuration files are stored in plain text. A user with access to the `/etc/tower/conf.d/` directory can view the passwords used to access the database. Access to the directories is controlled with permissions, so they are protected, but some security findings deem this protection to be inadequate. The solution is to encrypt the passwords individually.

Creating PostgreSQL password hashes

Supply the hash values that replace the plain text passwords within the automation controller configuration files.

Procedure

1. On your automation controller node, run the following:

```
# awx-manage shell_plus
```

2. Then run the following from the python prompt:

```
>>> from awx.main.utils import encrypt_value, get_encryption_key \
>>> postgres_secret = encrypt_value('$POSTGRES_PASS') \
>>> print(postgres_secret)
```

NOTE Replace the `$POSTGRES_PASS` variable with the actual plain text password you want to encrypt.

The output should resemble the following:

```
$encrypted$UTF8$AESCBC$Z0FBQUFBQmtLdGNRWXFjZGtKV1ZBR3hkNGVVbFFIU3hhY21UT081eXFkR09a
```

```
UWZLcG9TSmpndmZYQXFyRHVFQ3ZYSE15OUFuM1RHZHBqTFU3S0MyNEo2Y2JWUURSYktsdmc9PQ==
```

3. Copy the full values of these hashes and save them.

- The hash value begins with **\$encrypted\$**, and is not just the string of characters, as shown in the following example:

```
$encrypted$AESCBc$Z0FBQUFBQmNONU9BbGQ1VjJyNDJRVRTRKaFRIR09Ib2U5TGdaYVRfcXFXRjLmdm  
pZNjdoZVpEZ21QRWViMmNDOGJaM0dPeHN2b194NUxvQ1M5X3dSc1gxQ29TdDBKRkljWHc9PQ==
```

Note that the **\$*_PASS** values are already in plain text in your inventory file.

Encrypting the Postgres password

The following procedure replaces the plain text passwords with encrypted values. Perform the following steps on each node in the cluster:

Procedure

1. Edit **/etc/tower/conf.d/postgres.py** using:

```
$ vim /etc/tower/conf.d/postgres.py
```

2. Add the following line to the top of the file.

```
from awx.main.utils import decrypt_value, get_encryption_key
```

3. Remove the password value listed after 'PASSWORD': and replace it with the following line, replacing the supplied value of **\$encrypted..** with your own hash value:

```
decrypt_value(get_encryption_key('value'), '$encrypted$AESCBc$Z0FBQUFBQmNONU9BbGQ1Vj  
JyNDJRVRTRKaFRIR09Ib2U5TGdaYVRfcXFXRjLmdmpZNjdoZVpEZ21QRWViMmNDOGJaM0dPeHN2b194NUxvQ  
1M5X3dSc1gxQ29TdDBKRkljWHc9PQ=='),
```

NOTE The hash value in this step is the output value of **postgres_secret**.

4. The full **postgres.py** resembles the following:

```
# Ansible Automation platform controller database settings. from awx.main.utils  
import decrypt_value, get_encryption_key DATABASES = { 'default': {  
'ATOMIC_REQUESTS': True, 'ENGINE': 'django.db.backends.postgresql', 'NAME': 'awx',  
'USER': 'awx', 'PASSWORD':  
decrypt_value(get_encryption_key('value'), '$encrypted$AESCBc$Z0FBQUFBQmNONU9BbGQ1Vj  
JyNDJRVRTRKaFRIR09Ib2U5TGdaYVRfcXFXRjLmdmpZNjdoZVpEZ21QRWViMmNDOGJaM0dPeHN2b194NUxvQ  
1M5X3dSc1gxQ29TdDBKRkljWHc9PQ=='), 'HOST': '127.0.0.1', 'PORT': 5432, } }
```

3.7. Automation controller tuning

You can configure many automation controller settings by using the automation controller UI, API, and file based settings including:

- Live events in the automation controller UI
- Job event processing
- Control and execution node capacity
- Instance group and container group capacity
- Task management (job scheduling)
- Internal cluster routing
- Web server tuning

3.7.1. Managing live events in the automation controller UI

Events are sent to any node where there is a UI client subscribed to a job. This task is expensive, and becomes more expensive as the number of events that the cluster is producing increases and the number of control nodes increases, because all events are broadcast to all nodes regardless of how many clients are subscribed to particular jobs.

To reduce the overhead of displaying live events in the UI, administrators can choose to either:

- Disable live streaming events.
- Reduce the number of events shown per second or before truncating or hiding events in the UI.

When you disable live streaming of events, they are only loaded on hard refresh to a job's output detail page. When you reduce the number of events shown per second, this limits the overhead of showing live events, but still provides live updates in the UI without a hard refresh.

Disabling live streaming events

Procedure

1. Disable live streaming events by using one of the following methods:
 - a. In the API, set `UI_LIVE_UPDATES_ENABLED` to **False**.
 - b. Go to your automation controller. Open the **Miscellaneous System Settings** window. Set the **Enable Activity Stream** toggle to **Off**.

Settings to modify rate and size of events

If you cannot disable live streaming of events because of their size, reduce the number of events that are displayed in the UI. You can use the following settings to manage how many events are displayed:

Settings available for editing in the UI or API:

- `EVENT_STDOUT_MAX_BYTES_DISPLAY`: Maximum amount of `stdout` to display (as measured in bytes). This truncates the size displayed in the UI.
- `MAX_WEBSOCKET_EVENT_RATE`: Number of events to send to clients per second.

Settings available by using file based settings:

- `MAX_UI_JOB_EVENTS`: Number of events to display. This setting hides the rest of the events in the list.
- `MAX_EVENT_RES_DATA`: The maximum size of the ansible callback event's "res" data structure. The "res" is the full "result" of the module. When the maximum size of ansible callback events is reached, then the remaining output will be truncated. Default value is 700000 bytes.
- `LOCAL_STDOUT_EXPIRE_TIME`: The amount of time before a `stdout` file is expired and removed locally.

3.7.2. Settings for managing job event processing

The callback receiver processes all the output of jobs and writes this output as job events to the automation controller database. The callback receiver has a pool of workers that processes events in batches. The number of workers automatically increases with the number of CPU available on an instance.

Administrators can override the number of callback receiver workers with the setting `JOB_EVENT_WORKERS`. Do not set more than 1 worker per CPU, and there must be at least 1 worker. Greater values have more workers available to clear the Redis queue as events stream to the automation controller, but can compete with other processes such as the web server for CPU seconds, uses more database connections (1 per worker), and can reduce the batch size of events each worker commits.

Each worker builds up a buffer of events to write in a batch. The default amount of time to wait before writing a batch is 1 second. This is controlled by the `JOB_EVENT_BUFFER_SECONDS` setting. Increasing the amount of time the worker waits between batches can result in larger batch sizes.

3.7.3. Capacity settings for control and execution nodes

The following settings impact capacity calculations on the cluster. Set them to the same value on all control nodes by using the following file-based settings.

- `AWX_CONTROL_NODE_TASK_IMPACT`: Sets the impact of controlling jobs. You can use it when your control plane exceeds desired CPU or memory usage to control the number of jobs that your control plane can run at the same time.
- `SYSTEM_TASK_FORKS_CPU` and `SYSTEM_TASK_FORKS_MEM`: Influence how many resources are estimated to be consumed by each fork of Ansible. By default, 1 fork of Ansible is estimated to use 0.25 of a CPU and 100 Mb of memory.

3.7.4. Capacity settings for instance group and container group

Use the `max_concurrent_jobs` and `max_forks` settings available on instance groups to limit how many jobs and forks can be consumed across an instance group or container group.

- To calculate the `max_concurrent_jobs` you need on a container group consider the `pod_spec` setting for that container group. In the `pod_spec`, you can see the resource requests and limits for the automation job pod. Use the following equation to calculate the maximum concurrent jobs that you need:

$$\frac{((\text{number of worker nodes in kubernetes cluster}) * (\text{CPU available on each worker}))}{(\text{CPU request on pod_spec})} = \text{maximum number of concurrent jobs}$$

- For example, if your `pod_spec` indicates that a pod will request 250 mcpu Kubernetes cluster has 1 worker node with 2 CPU, the maximum number of jobs that you need to start with is 8.
 - You can also consider the memory consumption of the forks in the jobs. Calculate the appropriate setting of `max_forks` with the following equation:

$$\frac{((\text{number of worker nodes in kubernetes cluster}) * (\text{memory available on each worker}))}{(\text{memory request on pod_spec})} = \text{maximum number of forks}$$

- For example, given a single worker node with 8 GB of Memory, we determine that the `max_forks` we want to run is 81. This way, either 39 jobs with 1 fork can run (task impact is always forks + 1), or 2 jobs with forks set to 39 can run.
 - You might have other business requirements that motivate using `max_forks` or `max_concurrent_jobs` to limit the number of jobs launched in a container group.

3.7.5. Settings for scheduling jobs

The task manager periodically collects tasks that need to be scheduled and determines what instances have capacity and are eligible for running them. The task manager has the following workflow:

1. Find and assign the control and execution instances.
2. Update the job's status to waiting.
3. Message the control node through `pg_notify` for the dispatcher to pick up the task and start running it.

If the scheduling task is not completed within `TASK_MANAGER_TIMEOUT` seconds (default 300 seconds), the task is terminated early. Timeout issues generally arise when there are thousands of pending jobs.

One way the task manager limits how much work it can do in a single run is the `START_TASK_LIMIT` setting. This limits how many jobs it can start in a single run. The default is 100 jobs. If more jobs are pending, a new scheduler task is scheduled to run immediately after. Users who are willing to have potentially longer latency between when a job is launched and when it starts, to have greater overall throughput, can consider increasing the `START_TASK_LIMIT`. To see how long individual runs of the task manager take, use the Prometheus metric `task_manager__schedule_seconds`, available in </api/v2/metrics>.

Jobs elected to begin running by the task manager do not do so until the task manager process exits

and commits its changes. The `TASK_MANAGER_TIMEOUT` setting determines how long a single run of the task manager will run for before committing its changes. When the task manager reaches its timeout, it attempts to commit any progress it made. The task is not actually forced to exit until after a grace period (determined by `TASK_MANAGER_TIMEOUT_GRACE_PERIOD`) has passed.

3.7.6. Internal Cluster Routing

Automation controller cluster hosts communicate across the network within the cluster. In the inventory file for the traditional VM installer, you can indicate multiple routes to the cluster nodes that are used in different ways:

Example:

```
[automationcontroller]
controller1 ansible_user=ec2-user ansible_host=10.10.12.11 node_type=hybrid
routable_hostname=somehost.somecompany.org
```

- `controller1` is the inventory hostname for the automation controller host. The inventory hostname is what is shown as the instance hostname in the application. This can be useful when preparing for disaster recovery scenarios where you want to use the backup/restore method to restore the cluster to a new set of hosts that have different IP addresses. In this case you can have entries in `/etc/hosts` that map these inventory hostnames to IP addresses, and you can use internal IP addresses to mitigate any DNS issues when it comes to resolving public DNS names.
- `ansible_host=10.10.12.11` indicates how the installer reaches the host, which in this case is an internal IP address. This is not used outside of the installer.
- `routable_hostname=somehost.somecompany.org` indicates the hostname that is resolvable for the peers that connect to this node on the receptor mesh. Since it may cross multiple networks, we are using a hostname that will map to an IP address resolvable for the receptor peers.

3.7.7. Web server tuning

Control and Hybrid nodes each serve the UI and API of automation controller. WSGI traffic is served by the uwsgi web server on a local socket. ASGI traffic is served by Daphne. NGINX listens on port 443 and proxies traffic as needed.

To scale automation controller's web service, follow these best practices:

- Deploy multiple control nodes and use a load balancer to spread web requests over multiple servers.
- Set max connections per automation controller to 100.

To optimize automation controller's web service on the client side, follow these guidelines:

- Direct user to use dynamic inventory sources instead of individually creating inventory hosts by using the API.
- Use webhook notifications instead of polling for job status.

- Use the bulk APIs for host creation and job launching to batch requests.
- Use token authentication. For automation clients that must make many requests very quickly, using tokens is a best practice, because depending on the type of user, there might be additional overhead when using Basic authentication.

Additional resources

- [Scaling Automation Controller for API Driven Workloads](#)
- [Bulk API in Automation Controller](#)
- [Token-Based Authentication](#)

4. Management Jobs

Management Jobs assist in the cleaning of old data from automation controller, including system tracking information, tokens, job histories, and activity streams.

You can use this if you have specific retention policies or need to decrease the storage used by your automation controller database.

From the navigation panel, select **Automation Execution** › **Administration** › **Management Jobs**.

Management Jobs

Management Jobs assist in the cleaning of old data including system tracking information, tokens, job histories, and activity streams.

▼

Name

▼

Select name

Sort

Name

▼

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

328

329

330

331

332

333

334

335

336

337

338

339

340

341

342

343

344

345

346

347

348

349

350

351

352

353

354

355

356

357

358

359

360

361

362

363

364

365

366

367

368

369

370

371

372

373

374

375

376

377

378

379

380

381

382

383

384

385

386

387

388

389

390

391

392

393

394

395

396

397

398

399

400

401

402

403

404

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

429

430

431

432

433

434

435

436

437

438

439

440

441

442

443

444

445

446

447

448

449

450

451

452

453

454

455

456

457

458

459

460

461

462

463

464

465

466

467

468

469

470

471

472

473

474

475

476

477

478

479

480

481

482

483

484

485

486

487

488

489

490

491

492

493

494

495

496

497

498

499

500

501

502

503

504

505

506

507

508

509

510

511

512

513

514

515

516

517

518

519

520

521

522

523

524

525

526

527

528

529

530

531

532

533

534

535

536

537

538

539

540

541

542

543

544

545

546

547

548

549

550

551

552

553

554

555

556

557

558

559

560

561

562

563

564

565

566

567

568

569

570

571

572

573

574

575

576

577

578

579

580

581

582

583

584

585

586

587

588

589

590

591

592

593

594

595

596

597

598

599

600

601

602

603

604

605

606

607

608

609

610

611

612

613

614

615

616

617

618

619

620

621

622

623

624

625

626

627

628

629

630

631

632

633

634

635

636

637

638

639

640

641

642

643

644

645

646

647

648

649

650

651

652

653

654

655

656

657

658

659

660

661

662

663

664

665

666

667

668

669

670

671

672

673

674

675

676

677

678

679

680

681

682

683

684

685

686

687

688

689

690

691

692

693

694

695

696

697

698

699

700

701

702

703

704

705

706

707

708

709

710

711

712

713

714

715

716

717

718

719

720

721

722

723

724

725

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

756

757

758

759

760

761

762

763

764

765

766

767

768

769

770

771

772

773

774

775

776

777

778

779

780

781

782

783

784

785

786

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

803

804

805

806

807

808

809

810

811

812

813

814

815

816

817

818

819

820

821

822

823

824

825

826

827

828

829

830

831

832

833

834

835

836

837

838

839

840

841

842

843

844

845

846

847

848

849

850

851

852

853

854

855

856

857

858

859

860

861

862

863

864

865

866

867

868

869

870

871

872

873

874

875

876

877

878

879

880

881

882

883

884

885

886

887

888

889

890

891

892

893

894

895

896

897

898

899

900

901

902

903

904

905

906

907

908

909

910

911

912

913

914

915

916

917

918

919

920

921

922

923

924

925

926

927

928

929

930

931

932

933

934

935

936

937

938

939

940

941

942

943

944

945

946

947

948

949

950

951

952

953

954

955

956

957

958

959

960

961

962

963

964

965

966

967

968

969

970

971

972

973

974

975

976

977

978

979

980

981

982

983

984

985

986

987

988

989

990

991

992

993

994

995

996

997

998

999

1000

1001

1002

1003

1004

1005

1006

1007

1008

1009

1010

1011

1012

1013

1014

1015

1016

1017

1018

1019

1020

1021

1022

1023

1024

1025

1026

1027

1028

1029

1030

1031

1032

1033

1034

1035

1036

1037

1038

1039

1040

1041

1042

1043

1044

1045

1046

1047

1048

1049

1050

1051

1052

1053

1054

1055

1056

1057

1058

1059

1060

1061

1062

1063

1064

1065

1066

1067

1068

1069

1070

1071

1072

1073

1074

1075

1076

1077

1078

1079

1080

1081

1082

1083

1084

1085

1086

1087

1088

1089

1090

1091

1092

1093

1094

1095

1096

1097

1098

1099

1100

1101

1102

1103

1104

1105

1106

1107

1108

1109

1110

1111

1112

1113

1114

1115

1116

1117

1118

1119

1120

1121

1122

1123

1124

1125

1126

1127

1128

1129

1130

1131

1132

1133

1134

1135

1136

1137

1138

1139

1140

1141

1142

1143

1144

1145

1146

1147

1148

1149

1150

1151

1152

1153

1154

1155

1156

1157

1158

1159

1160

1161

1162

1163

1164

1165

1166

1167

1168

1169

1170

1171

1172

1173

1174

1175

1176

1177

1178

1179

1180

1181

1182

1183

1184

1185

1186

1187

1188

1189

1190

1191

1192

1193

1194

1195

1196

1197

1198

1199

1200

1201

1202

1203

1204

1205

1206

1207

1208

1209

1210

1211

1212

1213

1214

1215

1216

1217

1218

1219

1220

1221

1222

1223

1224

1225

1226

1227

1228

1229

1230

1231

1232

1233

1234

1235

1236

1237

1238

1239

1240

1241

1242

1243

1244

1245

1246

1247

1248

1249

1250

1251

1252

1253

1254

1255

1256

1257

1258

1259

1260

1261

1262

1263

1264

1265

1266

1267

1268

1269

1270

1271

1272

1273

1274

1275

1276

1277

1278

1279

1280

1281

1282

1283

1284

1285

1286

1287

1288

1289

1290

1291

1292

1293

1294

1295

1296

1297

1298

1299

1300

1301

1302

1303

1304

1305

1306

1307

1308

1309

1310

1311

1312

1313

1314

1315

1316

1317

1318

1319

1320

1321

1322

1323

1324

1325

1326

1327

1328

1329

1330

1331

1332

1333

1334

1335

1336

1337

1338

1339

1340

1341

1342

1343

1344

1345

1346

1347

1348

1349

1350

1351

1352

1353

1354

1355

1356

1357

1358

1359

1360

1361

1362

1363

1364

1365

1366

1367

1368

1369

1370

1371

1372

1373

1374

1375

1376

1377

1378

1379

1380

1381

1382

1383

1384

1385

1386

1387

1388

1389

1390

1391

1392

1393

1394

1395

1396

1397

1398

1399

1400

1401

1402

1403

1404

1405

1406

1407

1408

1409

1410

1411

1412

1413

1414

1415

1416

1417

1418

1419

1420

1421

1422

1423

1424

1425

1426

1427

1428

1429

1430

1431

1432

1433

1434

1435

1436

1437

1438

1439

1440

1441

1442

1443

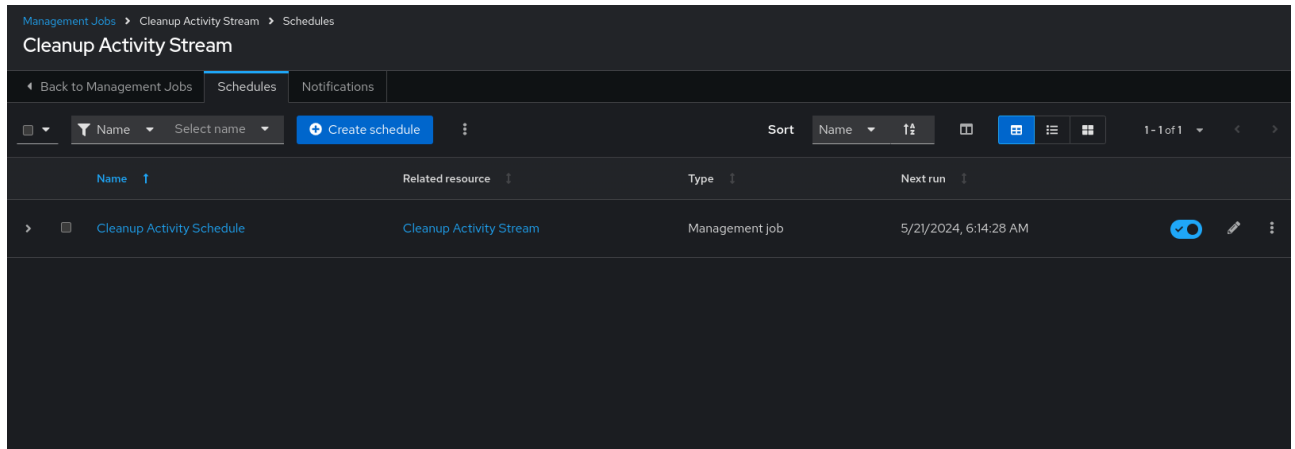
1444</

4.1.1. Scheduling deletion

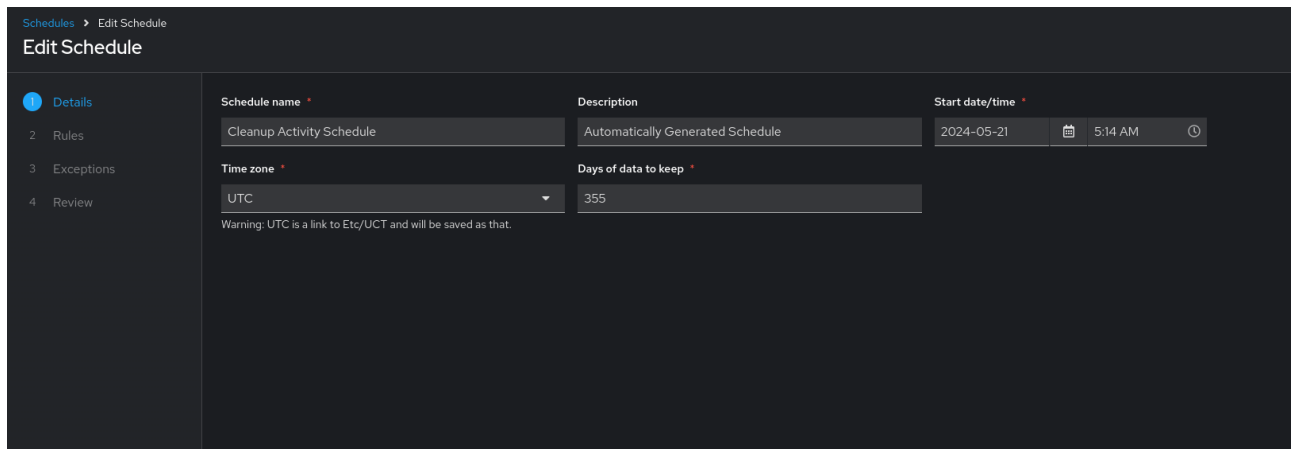
Use the following procedure to review or set a schedule for purging data marked for deletion:

Procedure

1. For a particular cleanup job, click the **Schedules** tab.



2. Click the name of the job, **Cleanup Activity Schedule** in this example, to review the schedule settings.
3. Click **[Edit schedule]** to change them. You can also click **[Create schedule]** to create a new schedule for this management job.



4. Enter the appropriate details into the following fields and click **Next**:
 - **Schedule name** required
 - **Start date/time** required
 - **Time zone** the entered Start Time should be in this time zone.
 - **Repeat frequency** the appropriate options display as the update frequency is modified including data you do not want to include by specifying exceptions.
 - **Days of data to keep** required - specify how much data you want to retain.

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

NOTE

Jobs are scheduled in UTC. Repeating jobs that run at a specific time of day can move relative to a local time zone when Daylight Saving Time shifts occur.

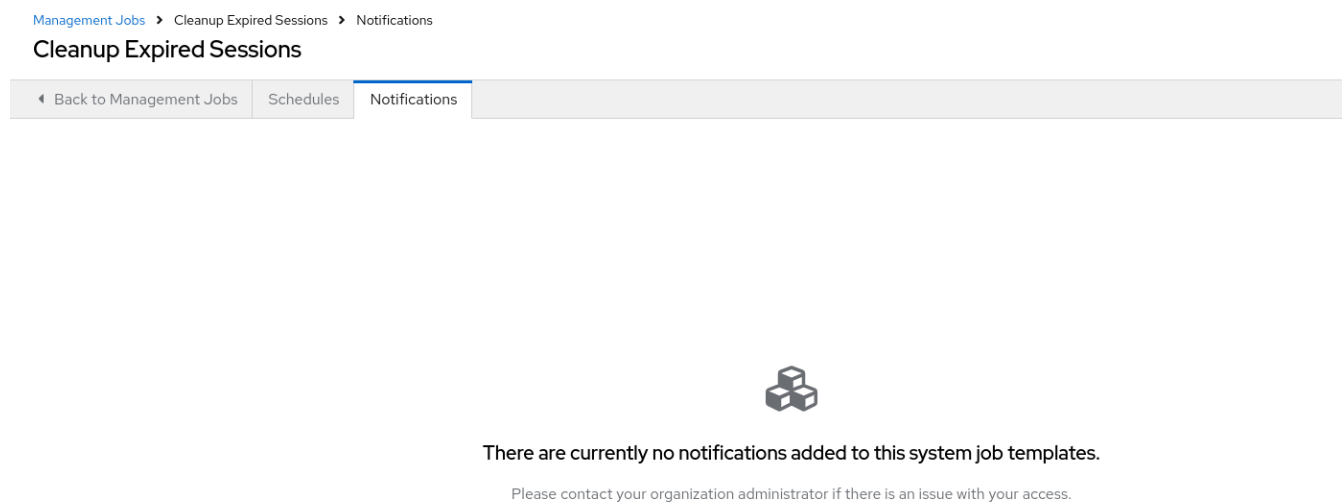
4.1.2. Setting notifications

Use the following procedure to review or set notifications associated with a management job:

Procedure

- For a particular cleanup job, select the **Notifications** tab.

If none exist, see [Creating a notification template](#) in *Using automation execution*.



4.1.3. Cleanup Expired Sessions


To remove expired sessions, click the launch  icon beside **Cleanup Expired Sessions**.

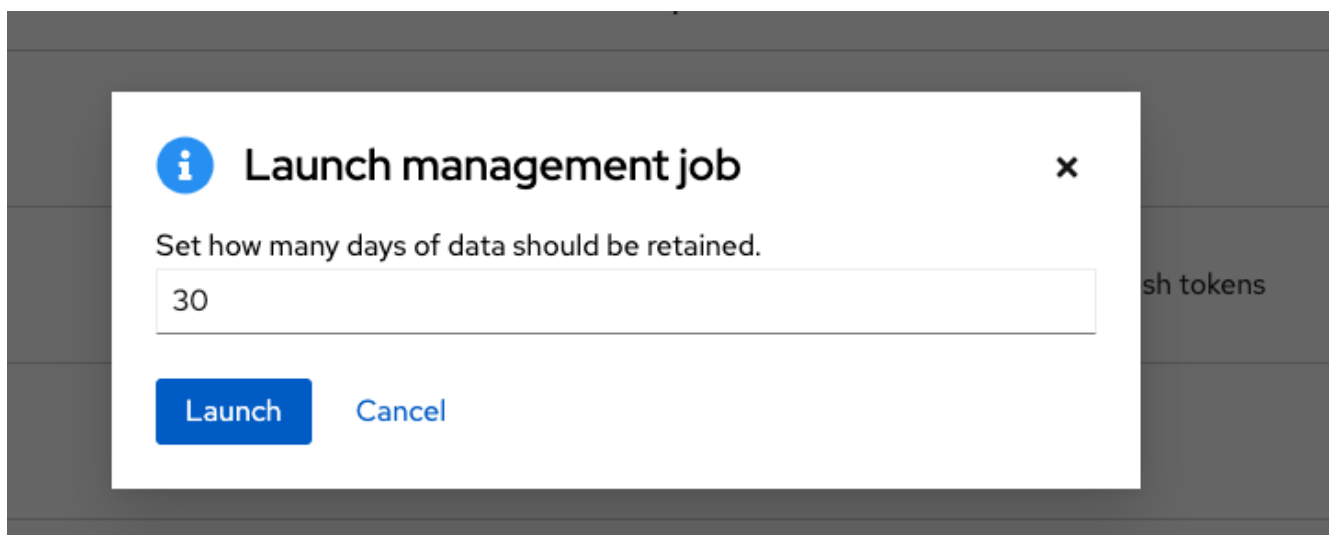
You can review or set a schedule for cleaning up expired sessions by performing the same procedure described for activity stream management jobs. For more information, see [Scheduling deletion](#).

You can also set or review notifications associated with this management job the same way as described in [Notifications](#) for activity stream management jobs.

For more information, see [Notifiers](#) in *Using automation execution*.

4.1.4. Removing Old Job History

To remove job history older than a specified number of days, click the launch  icon beside **Cleanup Job Details**.



Enter the number of days of data you want to save and click [**Launch**].

NOTE

The initial job run for an automation controller resource, such as Projects, or Job Templates, are excluded from **Cleanup Job Details**, regardless of retention value.

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

For more information, see [Scheduling deletion](#).

You can also set or review notifications associated with this management job in the same way as described in [Setting notifications](#) for activity stream management jobs, or for more information, see [link:https://docs.redhat.com/en/documentation/red_hat_automation_platform/2.5/html/using_automation_execution/controller-notificationsNotifiers](https://docs.redhat.com/en/documentation/red_hat_automation_platform/2.5/html/using_automation_execution/controller-notificationsNotifiers)].

5. Inventory File Importing

With automation controller you can select an inventory file from source control, rather than creating one from scratch. The files are non-editable, and as inventories are updated at the source, the inventories within the projects are also updated accordingly, including the `group_vars` and `host_vars` files or directory associated with them. SCM types can consume both inventory files and scripts. Both inventory files and custom inventory types use scripts.

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

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

Similarly, group descriptions also default to *imported*, but can also be overridden by `_awx_description`.

To use old inventory scripts in source control, see [Export old inventory scripts](#) in *Using automation execution*.

5.1. Source control management Inventory Source Fields

The source fields used are:

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

Additionally:

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

You can specify a location manually in the automation controller UI from the **Add source** page of an inventory. Refer to [Adding a source](#) for instructions on creating an inventory source.

When you update a project, refresh the listing to use the latest source control management (SCM) information. If no inventory sources use a project as an SCM inventory source, then the inventory listing might not be refreshed on update.

For inventories with SCM sources, the job **Details** page for inventory updates displays a status indicator for the project update and the name of the project.

The status indicator links to the project update job.

The project name links to the project.

You can perform an inventory update while a related job is running.

5.1.1. Supported File Syntax

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

IMPORTANT

You do not need to write inventory scripts in Python. You can enter any executable file in the source field and must run **chmod +x** for that file and check it into Git.

The following is a working example of JSON output that automation controller can read for the

import:

```
{
  "_meta": {
    "hostvars": {
      "host1": {
        "fly_rod": true
      }
    }
  },
  "all": {
    "children": [
      "groupA",
      "ungrouped"
    ]
  },
  "groupA": {
    "hosts": [
      "host1",
      "host10",
      "host11",
      "host12",
      "host13",
      "host14",
      "host15",
      "host16",
      "host17",
      "host18",
      "host19",
      "host2",
      "host20",
      "host21",
      "host22",
      "host23",
      "host24",
      "host25",
      "host3",
      "host4",
      "host5",
      "host6",
      "host7",
      "host8",
      "host9"
    ]
  }
}
```

Additional resources

- [test-playbooks/inventories](#)

- [inventories/changes.py](#)
- [How to migrate inventory scripts from Red Hat Ansible tower to Red Hat Ansible Automation Platform?](#)

6. Clustering

Clustering is sharing load between hosts. Each instance must be able to act as an entry point for UI and API access. This must enable the automation controller administrators to use load balancers in front of as many instances as they want and keep good data visibility.

NOTE Load balancing is optional, and it is entirely possible to have ingress on one or all instances as needed.

Each instance must be able to join the automation controller cluster and expand its ability to run jobs. This is a simple system where jobs can run anywhere rather than be directed on where to run. Also, you can group clustered instances into different pools or queues, called [Instance groups](#) as described in *Using automation execution*.

Ansible Automation Platform supports container-based clusters by using Kubernetes, meaning you can install new automation controller instances on this platform without any variation or diversion in functionality. You can create instance groups to point to a Kubernetes container. For more information, see the [Instance and container groups](#) section in *Using automation execution*.

Supported operating systems

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

- Red Hat Enterprise Linux 8 or later

NOTE Isolated instances are not supported in conjunction with running automation controller in OpenShift.

6.1. Setup considerations

Learn about the initial setup of clusters. To upgrade an existing cluster, see [Upgrade Planning](#) in the *Ansible Automation Platform Upgrade and Migration Guide*.

Note the following important considerations in the new clustering environment:

- PostgreSQL is a standalone instance and is not clustered. Automation controller does not manage replica configuration or database failover (if the user configures standby replicas).
- When you start a cluster, the database node must be a standalone server, and PostgreSQL must not be installed on one of the automation controller nodes.
- PgBouncer is not recommended for connection pooling with automation controller. Automation controller relies on `pg_notify` for sending messages across various components, and therefore, **PgBouncer** cannot readily be used in transaction pooling mode.

- All instances must be reachable from all other instances and they must be able to reach the database. It is also important for the hosts to have a stable address or hostname (depending on how the automation controller host is configured).
- All instances must be geographically collocated, with reliable low-latency connections between instances.
- To upgrade to a clustered environment, your primary instance must be part of the `default` group in the inventory and it needs to be the first host listed in the `default` group.
- Manual projects must be manually synced to all instances by the customer, and updated on all instances at once.
- The `inventory` file for platform deployments should be saved or persisted. If new instances are to be provisioned, the passwords, configuration options, and host names, must be made available to installation program.

6.2. Install and configure

Provisioning new instances for a VM-based install involves updating the `inventory` file and re-running the setup playbook. It is important that the inventory file has all passwords and information used when installing the cluster or other instances might be reconfigured. The inventory file has a single inventory group, `automationcontroller`.

NOTE

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

```
[automationcontroller]
hostA
hostB
hostC
[instance_group_east]
hostB
hostC
[instance_group_west]
hostC
hostD
```

NOTE

If no groups are selected for a resource, then the `automationcontroller` group is used, but if any other group is selected, then the `automationcontroller` group is not used in any way.

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

```
[automationcontroller]
hostA
hostB
```



```
hostC
[database]
hostDB
```

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

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

For more information about `routable_hostname`, see [General variables](#) in the *RPM installation*.

IMPORTANT

Earlier versions of automation controller used the variable name `rabbitmq_host`. If you are upgrading from an earlier version of the platform, and you previously specified `rabbitmq_host` in your inventory, rename `rabbitmq_host` to `routable_hostname` before upgrading.

6.2.1. Instances and ports used by automation controller and automation hub

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

- Port 80, 443 (normal automation controller and automation hub ports)
- Port 22 (SSH - ingress only required)
- Port 5432 (database instance - if the database is installed on an external instance, it must be opened to automation controller instances)

6.3. Status and monitoring by browser API

Automation controller reports as much status as it can using the browser API at `/api/v2/ping` to validate the health of the cluster. This includes the following:

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

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

6.4. Instance services and failure behavior

Each automation controller instance is made up of the following different services working collaboratively:

HTTP services

This includes the automation controller application itself and external web services.

Callback receiver

Receives job events from running Ansible jobs.

Dispatcher

The worker queue that processes and runs all jobs.

Redis

This key value store is used as a queue for event data propagated from ansible-playbook to the application.

Rsyslog

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

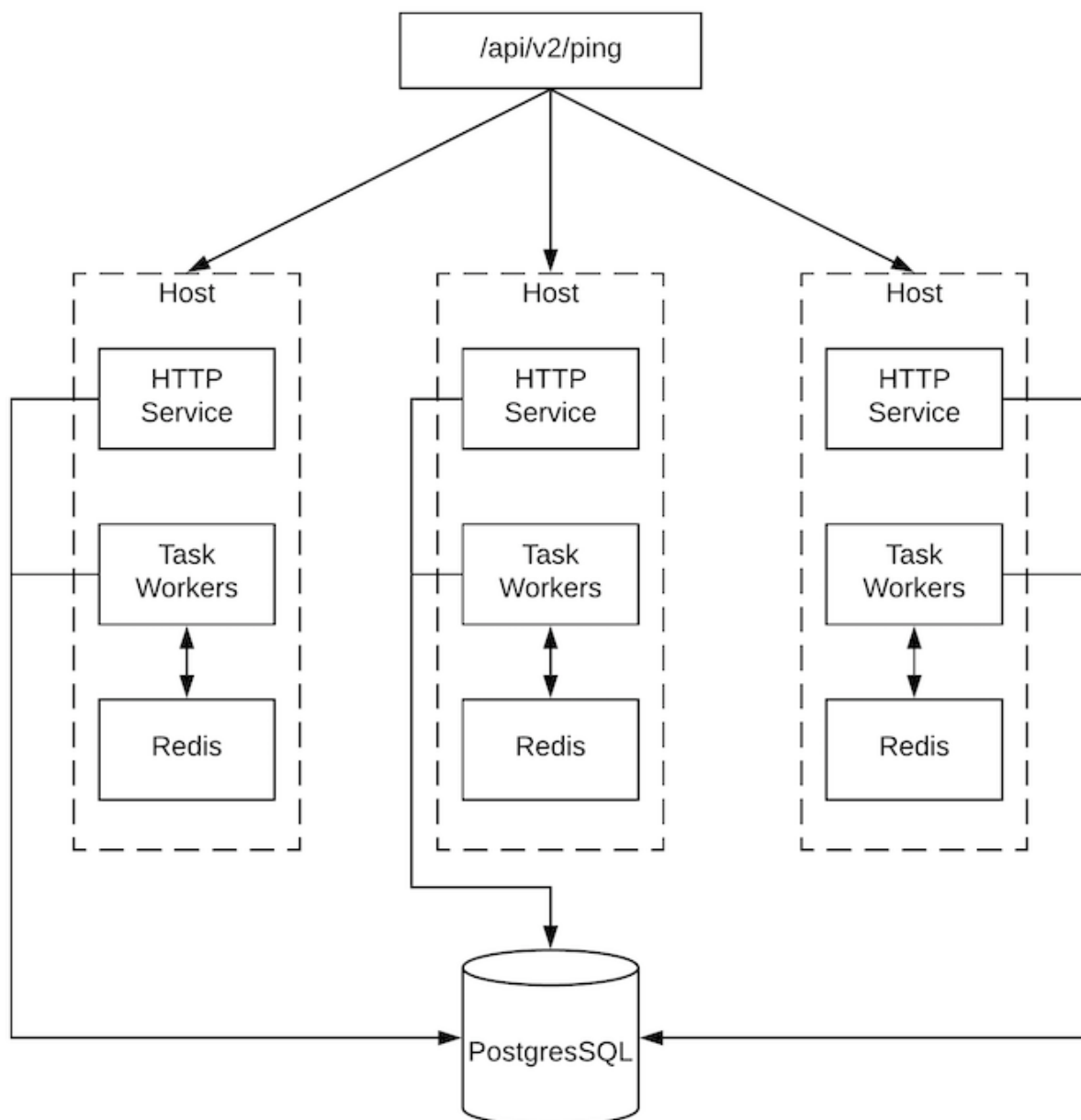
Automation controller is configured so that if any of these services or their components fail, then all services are restarted. If these fail often in a short span of time, then the entire instance is placed offline in an automated fashion to allow remediation without causing unexpected behavior.

For backing up and restoring a clustered environment, see the [Backup and restore clustered environments](#) section.

6.5. Job runtime behavior

The way jobs are run and reported to a *normal* user of automation controller does not change. On the system side, note the following differences:

- When a job is submitted from the API interface it is pushed into the dispatcher queue. Each automation controller instance connects to and receives jobs from that queue using a scheduling algorithm. Any instance in the cluster is just as likely to receive the work and run the task. If an instance fails while executing jobs, then the work is marked as permanently failed.



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

6.5.1. Job runs

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

instances from which a job runs on.

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

Troubleshooting

When you issue a **cancel** request on a running automation controller job, automation controller issues a **SIGINT** to the ansible-playbook process. While this causes Ansible to stop dispatching new tasks and exit, in many cases, module tasks that were already dispatched to remote hosts will run to completion. This behavior is similar to pressing **Ctrl-c** during a command-line Ansible run.

With respect to software dependencies, if a running job is canceled, the job is removed but the dependencies remain.

6.6. Deprovisioning instances

Re-running the setup playbook does not automatically deprovision instances since clusters do not currently distinguish between an instance that was taken offline intentionally or due to failure. Instead, shut down all services on the 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 command: **automation-controller-service stop**.
2. Run the following deprovision command from another instance to remove it from the automation controller cluster:

```
$ awx-manage deprovision_instance --hostname=<name used in inventory file>
```

The following is an example deprovision command:

```
$ awx-manage deprovision_instance --hostname=hostB
```

Deprovisioning instance groups in automation controller does not automatically deprovision or remove instance groups. For more information, see the [Deprovisioning instance groups](#) section in *Using automation execution*.

7. Implementing policy enforcement

Policy enforcement at automation runtime is a feature that uses encoded rules to define, manage, and enforce policies that govern how your users interact with your Ansible Automation Platform instance. Policy enforcement automates policy management, improving security, compliance, and efficiency.

OPA, or [Open Policy Agent](#), is a policy engine that offloads policy decisions from your Ansible instance. When it is triggered, the policy enforcement feature connects to OPA to retrieve policies specified in your configuration, and applies policy rules to your automation content. If OPA detects a policy violation, it will stop the action and give your user information about the policy violation.

Prerequisites

Before you can implement policy enforcement in your Ansible Automation Platform instance, you must have:

- Access to an OPA server that is reachable from your Ansible Automation Platform deployment.
- Configured Ansible Automation Platform with settings required for authenticating to your OPA server.
- Some familiarity with OPA and the Rego language, which is the language policies are written in.

For policy enforcement to work correctly, you must both configure the OPA server in your policy settings, and associate a specific policy with a particular resource. For example, a particular organization, inventory, or job template.

NOTE OPA API V1 is the only version currently supported in Ansible Automation Platform.

7.1. Configuring policy enforcement settings

You can specify how your Ansible Automation Platform instance interacts with OPA by modifying your global settings.

Prerequisites

- To configure policy enforcement, you must have administrative privileges.

NOTE If you do not configure the OPA server in your policy settings, policy evaluation will not occur when you run the job.

Procedure

1. From the navigation panel, select **Settings** › **Automation Execution** › **Policy**.
2. Click **Edit policy settings**.
3. On the Policy Settings page, fill out the following fields:

OPA Server hostname

Enter the name of the host that connects to the OPA service.

OPA server port

Enter the port that connects to the OPA service.

OPA authentication type

Select the OPA authentication type.

OPA custom authentication header

Enter a custom header to append to request headers for OPA authentication.

OPA request timeout

Enter the number of seconds until the connection times out.

OPA request retry count

Enter a figure for the number of times a request can attempt to connect to the OPA service before failing.

4. Depending on your authentication type, you might need to fill out the following fields.

a. If you selected Token as your authentication type:

OPA authentication token

Enter the OPA authentication token.

b. If you selected Certificate as your authentication type:

OPA client certificate content

Enter content of the CA certificate for mTLS authentication.

OPA client key content

Enter the client key for mTLS authentication.

OPA CA certificate content

Enter the content of the CA certificate for mTLS authentication.

5. Beneath the heading labeled **Options**:

Use SSL for OPA connection

Check this box to enable an SSL connection to the OPA service.

6. Click [**Save policy settings**].

7.2. Understanding OPA packages and rules

An OPA policy is organized in packages, which are namespaced collections of rules. The basic structure of an OPA policy looks like this:

```
package aap_policy_examples # Package name

import rego.v1 # Import required for Rego v1 syntax

# Rules define the policy logic
allowed := {
  "allowed": true,
  "violations": []
}
```

```
}
```

The key components of the rule's structure are:

Package declaration

This defines the namespace for your policy.

Rules

This defines the policy's logic and the decision that it returns.

These components together comprise the OPA policy name, which is formatted as `[package]/[rule]`. You will enter the OPA policy name when you configure enforcement points.

7.3. Configuring enforcement points

After you have set up your Ansible Automation Platform instance to communicate with the OPA server, you can set up enforcement points where you want the policy to be applied.

You can associate a policy with a job template, an inventory, or an organization. Enforcement then occurs in the following ways:

Organization

Jobs launched from a template owned by an organization will fail if the policy is violated. This configuration provides broad control over automation within organizational boundaries.

Inventory

Jobs using an inventory associated with a policy fail if the policy is violated. This configuration allows you to control access to specific infrastructure resources.

Job template

Jobs launched from a template associated with a policy fail if the job violates the associated policy. This configuration provides granular control over specific automation tasks.


NOTE

If you do not associate a policy with a resource, policy evaluation will not occur when you run the related job.

7.3.1. Associating a policy with an organization

To associate a policy with an organization, take the following steps.

Procedure


1. From the navigation panel, select **Access Management** › **Organizations**.
2. On the **Organizations** page:
 - a. To edit an existing organization, find the organization you want to edit and click the pencil icon  to go to the editing screen.
 - b. To create a new organization, click **[Create organization]**.

3. In the field labeled **Policy enforcement**, enter the query path associated with the policy you want to implement. You must format the query path as **package/rule**.
4. Click **[Next]** and then **[Finish]** to save your settings.

7.3.2. Associating a policy with an inventory

To associate a policy with an inventory, take the following steps:


Procedure

1. From the navigation panel, select **Automation Execution > Infrastructure > Inventories**.
2. On the **Inventories** page:
 - a. To edit an existing inventory, find the inventory you want to edit and click the pencil icon  to go to the editing screen.
 - b. To create a new inventory, click **[Create inventory]**.
3. In the field titled **Policy enforcement**, enter the query path associated with the policy you want to implement. You must format the query path as **package/rule**.
4. Click **[Save inventory]** if you are editing an existing inventory, or click **[Create inventory]** if you are creating a new inventory.

7.3.3. Associating a policy with a job template

To associate a policy with a job template, take the following steps:

Procedure

1. From the navigation panel, select **Automation Execution > Templates**.
2. On the **Automation Templates** page:
 - a. To edit an existing job template, find the job template you want to edit and click the pencil icon  to go to the editing screen.
 - b. To create a new job template, click **[Create template]**.
3. In the field titled **Policy enforcement**, enter the query path associated with the policy you want to implement. You must format the query path as **package/rule**.
4. Click **[Save job template]** if you are editing an existing job template, or click **[Create job template]** if you are creating a new job template.

7.4. Policy enforcement inputs and outputs

Use the following inputs and outputs to craft policies for use in policy enforcement.

Table 1. Input data

Input	Type	Description
id	Integer	The job's unique identifier.
name	String	Job template name.

Input	Type	Description
<code>created</code>	Datetime (ISO 8601)	Timestamp indicating when the job was created.
<code>created by</code>	Object	<p>Information about the user who created the job.</p> <ul style="list-style-type: none"> • <code>id</code>(integer): Unique identifier for the user • <code>username</code>(string): creator username • <code>is_superuser</code>(boolean): indicates if the user is a superuser
<code>credentials</code>	List of objects	<p>Credentials associated with job execution.</p> <ul style="list-style-type: none"> • <code>id</code> (integer): the credential's unique identifier • <code>name</code> (string): credential name • <code>description</code> (string): credential description • <code>organization</code> (integer or null): organization identifier associated with the credential • <code>credential_type</code> (integer): credential type identifier • <code>managed</code> (boolean): indicates if the credential is managed internally • <code>kind</code> (string): credential type (such as <code>ssh</code>, <code>cloud</code>, or <code>kubernetes</code>)

Input	Type	Description
<code>execution_environment</code>	Object	<p>Details about the execution environment used for the job.</p> <ul style="list-style-type: none"> • <code>id</code> (integer): the execution environment's unique identifier • <code>name</code> (string): execution environment name • <code>image</code> (string): container image used for execution • <code>pull</code> (string): pull policy for the execution environment
<code>extra_vars</code>	JSON	Extra variables provided for job execution.
<code>forks</code>	Integer	The number of parallel processes used for job execution.
<code>hosts_count</code>	Integer	The number of hosts targeted by the job.
<code>instance_group</code>	Object	<p>Information about the instance group handling the job, including:</p> <ul style="list-style-type: none"> • <code>id</code> (integer): the instance group's unique identifier • <code>name</code> (string): the instance group name • <code>capacity</code> (integer): the available capacity in the group • <code>jobs_running</code> (integer): the number of currently running jobs • <code>jobs_total</code> (integer): total jobs handled by the group • <code>max_concurrent_jobs</code> (integer): maximum concurrent jobs allowed • <code>max_forks</code> (integer): maximum forks allowed

Input	Type	Description
inventory	Object	<p>Inventory details used in the job execution, including:</p> <ul style="list-style-type: none"> • id (integer): the inventory's unique identifier • name (string): The inventory's name • description (string): inventory description • kind (string): the inventory type • total_hosts (integer): the total number of hosts in the inventory • total_groups (integer): the total number of groups in the inventory • has_inventory_sources (boolean): indicates if the inventory has external sources • total_inventory_sources (integer): the number of external inventory sources • has_active_failures (boolean): indicates if there are active failures in the inventory • hosts_with_active_failures (boolean): the number of hosts with active failures • inventory_sources (array): external inventory sources associated with the inventory

Input	Type	Description
<code>job_template</code>	Object	Information about the job template, including: <ul style="list-style-type: none"> • <code>id</code> (integer): the job template's unique identifier • <code>name</code> (string): the job template's name • <code>job_type</code> (string): type of job (for example, <code>run</code>)
<code>job_type</code>	Choice (String)	Type of job execution. Allowed values are: <ul style="list-style-type: none"> • <code>run</code> • <code>check</code> • <code>scan</code>
<code>job_type_name</code>	String	Human-readable name for the job type.
<code>labels</code>	List of objects	Labels associated with the job, including: <ul style="list-style-type: none"> • <code>id</code> (integer): the label's unique identifier • <code>name</code> (string): the label name • <code>organization</code> (object): the organization associated with the label <ul style="list-style-type: none"> ◦ <code>id</code> (integer): the organization's unique identifier ◦ <code>name</code> (string): the organization name

Input	Type	Description
launch_type	Choice (String)	How the job was launched. Allowed values include: <ul style="list-style-type: none"> • manual: manual • relaunch: relaunch • callback: callback • scheduled: scheduled • dependency: dependency • workflow: workflow • webhook: webhook • sync: sync • scm: SCM update
limit	String	The limit applied to the job execution.
launched_by	Object	Information about the user who launched the job, including: <ul style="list-style-type: none"> • id (integer): the user's unique identifier • name (string): the user name • type (type): the user type (for example, user or system) • url (string): the user's API URL
organization	Object	Information about the organization associated with the job, including: <ul style="list-style-type: none"> • id (integer): the organization's unique identifier • name (name): the organization's name
playbook	String	The playbook used in the job execution.

Input	Type	Description
<code>project</code>	Object	<p>Details about the project associated with the job, including:</p> <ul style="list-style-type: none"> • <code>id</code> (integer): the project's unique identifier • <code>name</code> (string): the project name • <code>status</code> (choice-string): the project status <ul style="list-style-type: none"> ◦ <code>successful</code>: Successful ◦ <code>failed</code>: failed ◦ <code>error</code>: error • <code>scm_type</code>(string): source control type (such as <code>git</code>, or <code>svn</code>) • <code>scm_url</code> (string): the source control repository URL • <code>scm_branch</code> (string): the branch used in the repository • <code>scm_refspec</code> (string): RefSpec for the repository • <code>scm_clean</code> (boolean): whether the SCM is cleaned before updates • <code>scm_track_submodules</code> (boolean): whether submodules are tracked • <code>scm_delete_on_update</code> (boolean): whether SCM deletes files on update
<code>scm_branch</code>	String	The specific branch to use for SCM.
<code>scm_revision</code>	String	SCM revision used for the job.
<code>workflow_job</code>	Object	Workflow job details, if the job is part of a workflow.
<code>workflow_job_template</code>	Object	Workflow job template details.

The following code block shows example input data from a demo job template launch:

```

{
  "id": 70,
  "name": "Demo Job Template",
  "created": "2025-03-19T19:07:03.329426Z",
  "created_by": {
    "id": 1,
    "username": "admin",
    "is_superuser": true,
    "teams": []
  },
  "credentials": [
    {
      "id": 3,
      "name": "Example Machine Credential",
      "description": "",
      "organization": null,
      "credential_type": 1,
      "managed": false,
      "kind": "ssh",
      "cloud": false,
      "kubernetes": false
    }
  ],
  "execution_environment": {
    "id": 2,
    "name": "Default execution environment",
    "image": "registry.redhat.io/ansible-automation-platform-25/ee-supported-
rhel8@sha256:b9f60d9ebbbb5fdc394186574b95dea5763b045ceff253815afeb435c626914d",
    "pull": ""
  },
  "extra_vars": {
    "example": "value"
  },
  "forks": 0,
  "hosts_count": 0,
  "instance_group": {
    "id": 2,
    "name": "default",
    "capacity": 0,
    "jobs_running": 1,
    "jobs_total": 38,
    "max_concurrent_jobs": 0,
    "max_forks": 0
  },
  "inventory": {
    "id": 1,
    "name": "Demo Inventory",
    "description": "",
    "kind": "",
    "total_hosts": 1,

```

```

    "total_groups": 0,
    "has_inventory_sources": false,
    "total_inventory_sources": 0,
    "has_active_failures": false,
    "hosts_with_active_failures": 0,
    "inventory_sources": []
  },
  "job_template": {
    "id": 7,
    "name": "Demo Job Template",
    "job_type": "run"
  },
  "job_type": "run",
  "job_type_name": "job",
  "labels": [
    {
      "id": 1,
      "name": "Demo label",
      "organization": {
        "id": 1,
        "name": "Default"
      }
    }
  ],
  "launch_type": "workflow",
  "limit": "",
  "launched_by": {
    "id": 1,
    "name": "admin",
    "type": "user",
    "url": "/api/v2/users/1/"
  },
  "organization": {
    "id": 1,
    "name": "Default"
  },
  "playbook": "hello_world.yml",
  "project": {
    "id": 6,
    "name": "Demo Project",
    "status": "successful",
    "scm_type": "git",
    "scm_url": "https://github.com/ansible/ansible-tower-samples",
    "scm_branch": "",
    "scm_refspec": "",
    "scm_clean": false,
    "scm_track_submodules": false,
    "scm_delete_on_update": false
  },
  "scm_branch": "",
  "scm_revision": "",

```



```

"workflow_job": {
  "id": 69,
  "name": "Demo Workflow"
},
"workflow_job_template": {
  "id": 10,
  "name": "Demo Workflow",
  "job_type": null
}
}

```

Table 2. Output data

Input	Type	Description
<code>allowed</code>	Boolean	Indicates whether the action is permitted
<code>violations</code>	List of strings	Reasons why the action is not permitted

The following code block shows an example of expected output from the OPA policy query:

```

{
  "allowed": false,
  "violations": [
    "No job execution is allowed",
    ...
  ],
  ...
}

```

8. Automation controller logs

Automation controller logs are accessed in different ways depending on whether you have an RPM-based or containerized installation of Ansible Automation Platform.

8.1. Accessing automation controller logs for containerized Ansible Automation Platform

Logs for containerized Ansible Automation Platform are not saved to specific files. The application logs are sent to the container `stdout` and handled by Podman with `journald`.

The three containers associated with automation controller are:

- `automation-controller-rsyslog`
- `automation-controller-task`

- `automation-controller-web`

For more information about the purpose of each of these containers and how to inspect the logs, see [Diagnosing the problem](#) in *Containerized installation*.

8.2. Accessing automation controller logs for RPM-based Ansible Automation Platform

Automation controller logfiles can be accessed from two centralized locations:

- `/var/log/tower/`
- `/var/log/supervisor/`

In the `/var/log/tower/` directory, you can view logfiles captured by:

- **tower.log:** Captures the log messages such as runtime errors that occur when the job is executed.
- **callback_receiver.log:** Captures callback receiver logs that handles callback events when running ansible jobs.
- **dispatcher.log:** Captures log messages for the automation controller dispatcher worker service.
- **job_lifecycle.log:** Captures details of the job run, whether it is blocked, and what condition is blocking it.
- **management_playbooks.log:** Captures the logs of management playbook runs, and isolated job runs such as copying the metadata.
- **rsyslog.err:** Captures rsyslog errors authenticating with external logging services when sending logs to them.
- **task_system.log:** Captures the logs of tasks that automation controller is running in the background, such as adding cluster instances and logs related to information gathering or processing for analytics.
- **tower_rbac_migrations.log:** Captures the logs for rbac database migration or upgrade.
- **tower_system_tracking_migrations.log:** Captures the logs of the controller system tracking migration or upgrade.
- **wsbroadcast.log:** Captures the logs of websocket connections in the controller nodes.

In the `/var/log/supervisor/` directory, you can view logfiles captured by:

- **awx-callback-receiver.log:** Captures the log of callback receiver that handles callback events when running ansible jobs, managed by `supervisord`.
- **awx-daphne.log:** Captures the logs of Websocket communication of WebUI.
- **awx-dispatcher.log:** Captures the logs that occur when dispatching a task to an automation controller instance, such as when running a job.
- **awx-rsyslog.log:** Captures the logs for the `rsyslog` service.
- **awx-uwsgi.log:** Captures the logs related to uWSGI, which is an application server.

- **awx-wsbroadcast.log:** Captures the logs of the websocket service that is used by automation controller.
- **failure-event-handler.stderr.log:** Captures the standard errors for `/usr/bin/failure-event-handler` supervisord's subprocess.
- **supervisord.log:** Captures the logs related to `supervisord` itself.
- **wsrelay.log:** Captures the communication logs within the websocket relay server.
- **ws_heartbeat.log:** Captures the periodic checks on the health of services running on the host.
- **rsyslog_configurer.log:** Captures rsyslog configuration activity associated with authenticating with external logging services.

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

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

- `/var/log/nginx/`
- `/var/lib/pgsql/data/pg_log/`
- `/var/log/redis/`

Troubleshooting

Error logs can be found in the following locations:

- Automation controller server errors are logged in `/var/log/tower`.
- Supervisors logs can be found in `/var/log/supervisor/`.
- Nginx web server errors are logged in the httpd error log.
- Configure other automation controller logging needs in `/etc/tower/conf.d/`.

Explore client-side issues using the JavaScript console built into most browsers and report any errors to Ansible through the Red Hat Customer portal at: <https://access.redhat.com/>.

9. Logging and Aggregation

Logging provides the capability to send detailed logs to third-party external log aggregation services. Services connected to this data feed serve as a means of gaining insight into automation controller use or technical trends. The data can be used to analyze events in the infrastructure, monitor for anomalies, and correlate events in one service with events in another.

The types of data that are most useful to automation controller are job fact data, job events or job runs, activity stream data, and log messages. The data is sent in JSON format over a HTTP connection using minimal service-specific adjustments engineered in a custom handler or through an imported library.

The version of `rsyslog` that is installed by automation controller does not include the following `rsyslog` modules:

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

After installing automation controller, you must only use the automation controller provided `rsyslog` package for any logging outside of automation controller that might have previously been done with the RHEL provided `rsyslog` package.

If you already use `rsyslog` for logging system logs on the automation controller instances, you can continue to use `rsyslog` to handle logs from outside of automation controller by running a separate `rsyslog` process (using the same version of `rsyslog` that automation controller uses), and pointing it to a separate `/etc/rsyslog.conf` file.

Use the `/api/v2/settings/logging/` endpoint to configure how the automation controller `rsyslog` process handles messages that have not yet been sent in the event that your external logger goes offline:

- `LOG_AGGREGATOR_ACTION_MAX_DISK_USAGE_GB`: Maximum disk persistence for `rsyslogd` action queuing in GB.

Specifies the amount of data to store (in gigabytes) during an outage of the external log aggregator (defaults to 1).

Equivalent to the `rsyslogd queue.maxDiskSpace` setting.

- `LOG_AGGREGATOR_ACTION_QUEUE_SIZE`: Maximum number of messages that can be stored in the log action queue.

Defines how large the `rsyslog` action queue can grow in number of messages stored. This can have an impact on memory use. When the queue reaches 75% of this number, the queue starts writing to disk (`queue.highWatermark` in `rsyslog`). When it reaches 90%, `NOTICE`, `INFO`, and `DEBUG` messages start to be discarded (`queue.discardMark` with `'queue.discardSeverity=5'`).

Equivalent to the `rsyslogd queue.size` setting on the action.

It stores files in the directory specified by `LOG_AGGREGATOR_MAX_DISK_USAGE_PATH`.

- `LOG_AGGREGATOR_MAX_DISK_USAGE_PATH`: Specifies the location to store logs that should be retried after an outage of the external log aggregator (defaults to `/var/lib/awx`). Equivalent to the `rsyslogd queue.spoolDirectory` setting.

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

9.1. Loggers

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

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

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

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

From the navigation panel, select **Settings › Automation Execution › Logging** to configure the logging components in automation controller.

9.1.1. Log message schema

Common schema for all loggers:

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

9.1.2. Activity stream schema

This uses the fields common to all loggers listed in [Log message schema](#).

It has the following additional fields:

- **actor**: Username of the user who took the action documented in the log.
- **changes**: JSON summary of what fields changed, and their old or new values.
- **operation**: The basic category of the changes logged in the activity stream, for instance, "associate".
- **object1**: Information about the primary object being operated on, consistent with what is shown in the activity stream.
- **object2**: If applicable, the second object involved in the action.

This logger reflects the data being saved into job events, except when they would otherwise conflict

with expected standard fields from the logger, in which case the fields are nested. Notably, the field `host` on the `job_event` model is given as `event_host`. There is also a sub-dictionary field, `event_data` within the payload, which contains different fields depending on the specifics of the Ansible event.

This logger also includes the common fields in [Log message schema](#).

9.1.3. Scan / fact / system tracking data schema

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

- **services**: For services scans, this field is included and has keys based on the name of the service.

NOTE

Periods are not allowed by elastic search in names, and are replaced with "_" by the log formatter.

- **package**: Included for log messages from package scans.
- **files**: Included for log messages from file scans.
- **host**: Name of the host the scan applies to.
- **inventory_id**: The inventory id the host is inside of.

This logger also includes the common fields in [Log message schema](#).

9.1.4. Job status changes

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

This logger also includes the common fields in [Log message schema](#) and fields present on the job model.

9.1.5. Automation controller logs

This logger also includes the common fields in [Log message schema](#).

In addition, this contains a `msg` field with the log message. Errors contain a separate `traceback` field. From the navigation panel, select **Settings** › **Automation Execution** › **Logging**. On the **Logging Settings** page click **[Edit]** and use the **ENABLE EXTERNAL LOGGING** option to enable or disable the logging components.

9.1.6. Logging Aggregator Services

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

- [Splunk](#)
- [Loggly](#)
- [Sumologic](#)
- [Elastic Stack \(formerly ELK stack\)](#)

Splunk

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

```
https://<yourcontrollerfqdn>/api/v2/settings/logging

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

NOTE

The Splunk HTTP Event Collector listens on port 8088 by default, so you must provide the full HEC event URL (with the port number) for **LOG_AGGREGATOR_HOST** for incoming requests to be processed successfully.

Typical values are shown in the following example:

The screenshot shows the 'Logging Settings' configuration page. It includes fields for 'Logging Aggregator' (URL), 'Logging Aggregator Port' (default -1), 'Logging Aggregator Type' (dropdown set to 'splunk'), 'Logging Aggregator Username', and 'Logging Aggregator Password/Token'. A section titled 'Loggers Sending Data to Log Aggregator Form' shows a list of loggers: 'awx', 'activity_stream', 'job_events', 'system_tracking', and 'broadcast_websocket'. Other settings include 'Cluster-wide unique identifier', 'Logging Aggregator Protocol' (dropdown set to 'HTTPS/HTTP'), 'TCP Connection Timeout' (set to 5), 'Logging Aggregator Level Threshold' (dropdown set to 'INFO'), 'Maximum number of messages that can be stored in the log action queue' (set to 131072), 'Maximum disk persistence for rsyslogd action queuing (in GB)' (set to 1), 'File system location for rsyslogd disk persistence' (set to '/var/lib/awx'), 'Log Format For API 4XX Errors' (set to 'status {status_code} received by user {user_name} attempting to ac...'), and 'Options' (checkboxes for 'Log System Tracking Facts Individually', 'Enable External Logging', and 'Enable/disable HTTPS certificate verification' which is checked).

For more information on configuring the HTTP Event Collector, see the [Splunk documentation](#).

Loggly

For more information on sending logs through Loggly's HTTP endpoint, see the [Loggly documentation](#).

Loggly uses the URL convention shown in the **Logging Aggregator** field in the following example:

Logging Aggregator ?

Revert

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

Sumologic

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

The screenshot displays the Sumologic web interface. At the top, there is a navigation bar with links for Library, Search, Metrics, Dashboards, Manage, and Help. The user's name, Alan (Re), is visible in the top right corner.

The main section is titled "Unnamed Search". It contains a search query box with the following JSON-based search criteria:

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

Below the query box, there are options to select a time range (Last 15 Minutes) and a button to "Start". A checkbox for "Use Receipt Time" is also present.

The search results are displayed in a timeline view. The timeline shows a single event at 11/30/2016 2:58:21 PM -0500. The event is represented by a blue bar on the timeline.

Below the timeline, there is a "Messages" section. It includes a "Display Fields" list with "Time", "actor", "message2", "object1", and "Message". The "actor" field is selected. The "Hidden Fields" list includes "Collector", "Size", "Source", "Source Category", "Source Host", and "Source Name".

A modal window is open, showing the "actor" field details. It includes a "DISPLAY" toggle, a table of values, and a "DRILLDOWN" section.

VALUES	#	%
admin	2	100.00%

The "DRILLDOWN" section includes links for "Top Values", "Top Values Over Time", and "Bottom Values".

The main log entry is displayed below the modal window. It shows the following details:

- Host: 207.67.11.130
- Name: Http Input
- Category: Http Input
- Timestamp: 11/30/2016 15:07:39.883 -0500
- Actor: admin
- Message: Activity Stream update entry for project
- Object1: "project"
- Object2: ""
- Actor: "admin"
- Type: "Logstash"

The log entry is expanded, showing the full JSON message:

```
{
  "cluster_host_id": "tower",
  "relationship": "",
  "tags": [ ],
  "@timestamp": "2016-11-30T20:07:39.883Z",
  "object1": "setting",
  "object2": "",
  "actor": "admin",
  "type": "Logstash",
  "level": "INFO",
  "@version": "1",
  "changes": "{\\\"name\\\": [\\\"AlanCoding exampleszzsafasdfqoqtz\\\"], \\\"AlanCoding exampleszzsafasdfqoqtz\\\"}",
  "operation": "update",
  "message": "Activity Stream update entry for project",
  "path": "./awx/main/middleware.py",
  "logger_name": "awx.analytics.activity_stream",
  "host": "tower",
  "object1": "project",
  "type": "Logstash"
}
```


Elastic stack (formerly ELK stack)

If you are setting up your own version of the elastic stack, the only change you require is to add the following lines to the logstash `logstash.conf` file:

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

NOTE

Backward-incompatible changes were introduced with Elastic 5.0.0, and different configurations might be required depending on what version you are using.

9.2. Setting up logging

Use the following procedure to set up logging to any of the aggregator types.

Procedure

1. From the navigation panel, select **Settings** › **Automation Execution** › **Logging**.
2. On the **Logging settings** page, click [**Edit**].

Logging Settings

Logging Aggregator [ⓘ] Logging Aggregator Port [ⓘ] Logging Aggregator Type [ⓘ]

Logging Aggregator Username [ⓘ] Logging Aggregator Password/Token [ⓘ]

▼ Loggers Sending Data to Log Aggregator Form [ⓘ] YAML JSON

- awx
- activity_stream
- job_events
- system_tracking
- broadcast_websocket

Cluster-wide unique identifier. [ⓘ] Logging Aggregator Protocol [ⓘ] TCP Connection Timeout [ⓘ]

Logging Aggregator Level Threshold [ⓘ] Maximum number of messages that can be stored in the log action queue [ⓘ] Maximum disk persistence for rsyslogd action queuing (in GB) [ⓘ]

File system location for rsyslogd disk persistence [ⓘ] Log Format For API 4XX Errors [ⓘ] Options

/var/lib/awx status {status_code} received by user {user_name} attempting to ac... ☐ Log System Tracking Facts Individually [ⓘ]

☐ Enable External Logging [ⓘ] ☒ Enable/disable HTTPS certificate verification [ⓘ] ☐ Enable rsyslogd debugging [ⓘ]

Save **Cancel**

3. You can configure the following options:

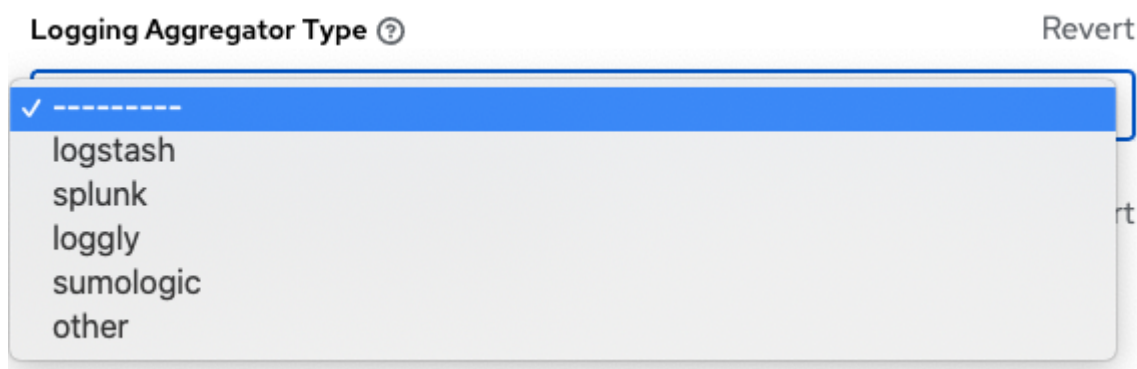
- **Logging Aggregator:** Enter the hostname or IP address that you want to send logs to.
- **Logging Aggregator Port:** Specify the port for the aggregator if it requires one.


NOTE

When the connection type is HTTPS, you can enter the hostname as a URL with a port number, after which, you are not required to enter the port again. However, TCP and UDP connections are determined by the hostname and port number combination, rather than URL. Therefore, in the case of a

TCP or UDP connection, supply the port in the specified field. If a URL is entered in the **Logging Aggregator** field instead, its hostname portion is extracted as the hostname.

- **Logging Aggregator Type:** Click to select the aggregator service from the list:



- **Logging Aggregator Username:** Enter the username of the logging aggregator if required.
- **Logging Aggregator Password/Token:** Enter the password of the logging aggregator if required.
- **Loggers to Send Data to the Log Aggregator Form:** All four types of data are pre-populated by default. Click the tooltip  icon next to the field for additional information on each data type. Delete the data types you do not want.
- **Cluster wide unique identifier:** Use this to uniquely identify instances.
- **Logging Aggregator Protocol:** Click to select a connection type (protocol) to communicate with the log aggregator. Subsequent options vary depending on the selected protocol.
- **TCP Connection Timeout:** Specify the connection timeout in seconds. This option is only applicable to HTTPS and TCP log aggregator protocols.
- **Logging Aggregator Level Threshold:** Select the level of severity you want the log handler to report.
- **Maximum number of messages that can be stored in the log action queue:** Defines how large the `rsyslog` action queue can grow in number of messages stored. This can have an impact on memory use. When the queue reaches 75% of this number, the queue starts writing to disk (`queue.highWatermark` in `rsyslog`). When it reaches 90%, `NOTICE`, `INFO`, and `DEBUG` messages start to be discarded (`queue.discardMark` with `queue.discardSeverity=5`).
- **Maximum disk persistence for rsyslogd action queuing (in GB):** The amount of data to store (in gigabytes) if an `rsyslog` action takes time to process an incoming message (defaults to 1). Equivalent to the `rsyslogd queue.maxdiskspace` setting on the action (e.g. `omhttp`). It stores files in the directory specified by `LOG_AGGREGATOR_MAX_DISK_USAGE_PATH`.
- **File system location for rsyslogd disk persistence:** Location to persist logs that should be retried after an outage of the external log aggregator (defaults to `/var/lib/awx`). Equivalent to the `rsyslogd queue.spoolDirectory` setting.
- **Log Format For API 4XX Errors:** Configure a specific error message. For more information, see [API 4XX Error Configuration](#).

Set the following options:

- **Log System Tracking Facts Individually:** Click the tooltip ⓘ icon for additional information, such as whether or not you want to turn it on, or leave it off by default.

1. Review your entries for your chosen logging aggregation. The following example is set up for Splunk:

Settings > Logging

Edit Details ⓘ

Enable External Logging ⓘ ☒ On [Revert](#)

Logging Aggregator Type ⓘ [Revert](#) splunk

Log System Tracking Facts Individually ⓘ ☐ Off [Revert](#)

TCP Connection Timeout ⓘ [Revert](#) 5

Logging Aggregator ⓘ [Revert](#) 172.16.185.132

Logging Aggregator Username ⓘ [Revert](#)

Logging Aggregator Password/Token ⓘ [Revert](#)

Logging Aggregator Port ⓘ [Revert](#) 80

Logging Aggregator Protocol ⓘ [Revert](#) HTTPS/HTTP

Logging Aggregator Level Threshold ⓘ [Revert](#) INFO

Enable/disable HTTPS certificate verification ⓘ [Revert](#) ☒ On

Loggers Sending Data to Log Aggregator Form ⓘ [Revert](#)

```
1 [
2   "awx",
3   "activity_stream",
4   "job_events",
5   "system_tracking"
6 ]
```

Log Format For API 4XX Errors ⓘ [Revert](#)

status {status_code} received by user {user_name} attempting to access {url_path} from {remote_addr}

[Save](#) [Revert all to default](#) [Cancel](#)

- **Enable External Logging:** Select this checkbox if you want to send logs to an external log aggregator.
- **Enable/disable HTTPS certificate verification:** Certificate verification is enabled by default for the HTTPS log protocol. Select this checkbox if you want the log handler to verify the HTTPS certificate sent by the external log aggregator before establishing a connection.
- **Enable rsyslogd debugging:** Select this checkbox to enable high verbosity debugging for **rsyslogd**. Useful for debugging connection issues for external log aggregation.

1. Click **[Save]** or **[Cancel]** to abandon the changes.

9.3. API 4XX Error Configuration

When the API encounters an issue with a request, it typically returns an HTTP error code in the 400 range along with an error. When this happens, an error message is generated in the log that follows the following pattern:

```
' status {status_code} received by user {user_name} attempting to access {url_path}
from {remote_addr} '
```

These messages can be configured as required. Use the following procedure to modify the default API 4XX errors log message format.

Procedure

1. From the navigation panel, select **Settings › Automation Execution › Logging**.
2. On the **Logging settings** page, click **[Edit]**.
3. Modify the field **Log Format For API 4XX Errors**.

Items surrounded by `{ }` are substituted when the log error is generated. The following variables can be used:

- **status_code**: The HTTP status code the API is returning.
- **user_name**: The name of the user that was authenticated when making the API request.
- **url_path**: The path portion of the URL being called (the API endpoint).
- **remote_addr**: The remote address received by automation controller.
- **error**: The error message returned by the API or, if no error is specified, the HTTP status as text.

9.4. Troubleshooting logging

Logging Aggregation

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

API 4XX Errors

You can include the API error message for 4XX errors by modifying the log format for those messages. Refer to the [API 4XX Error Configuration](#).

LDAP

You can enable logging messages for the LDAP adapter. For more information, see [API 4XX Error Configuration](#).

SAML

You can enable logging messages for the SAML adapter the same way you can enable logging for LDAP.

10. Metrics

A metrics endpoint, `/api/controller/v2/metrics/` is available in the API that produces instantaneous metrics about automation controller, which can be consumed by system monitoring software such as the open source project Prometheus.

The types of data shown at the `metrics/` endpoint are `Content-type: text/plain` and `application/json`.

This endpoint has useful information, such as counts of how many active user sessions there are, or

how many jobs are actively running on each automation controller node.

You can configure Prometheus to scrape these metrics from automation controller by hitting the automation controller metrics endpoint and storing this data in a time-series database.

Clients can later use Prometheus in conjunction with other software such as Grafana or Metricbeat to visualize that data and set up alerts.

10.1. Setting up Prometheus

To set up and use Prometheus, you must install Prometheus on a virtual machine or container.

For more information, see the [First steps with Prometheus](#) documentation.

Procedure

1. In the Prometheus configuration file (typically `prometheus.yml`), specify a `<token_value>`, a valid username and password for an automation controller user that you have created, and a `<controller_host>`.

NOTE

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

Using an OAuth2 Token, created at the `/api/v2/tokens` endpoint to authenticate Prometheus with automation controller, the following example provides a valid scrape configuration if the URL for your automation controller's metrics endpoint is `/https://controller_host:443/metrics`.

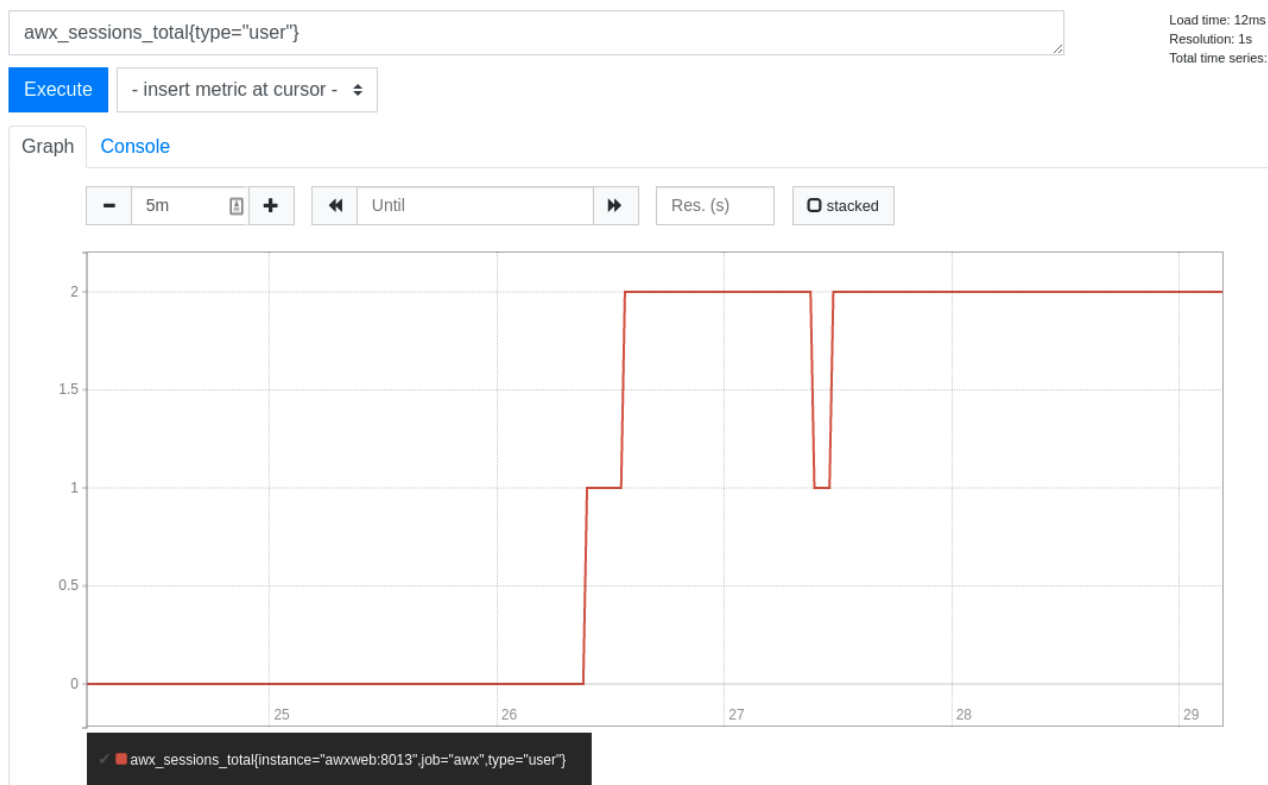
```
scrape_configs

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

For help configuring other aspects of Prometheus, such as alerts and service discovery configurations, see the [Prometheus configuration](#) documentation.

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

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



Next steps

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

11. Subscription management in Ansible Automation Platform and automation controller

Ansible Automation Platform provides capabilities to monitor usage, activate subscriptions, and maintain compliance with Red Hat subscription requirements.

11.1. Host metrics and subscription

Host metrics can be used to accurately count node usage and ensure subscription compliance. For example, if a host is no longer in use or otherwise should not be counted towards the subscription total, it can be soft-deleted.

11.1.1. Host Metrics dashboard

To view your host metrics, in the navigation pane, select **Automation Analytics** > **Host Metrics**.

The Host Metrics dashboard provides high level automation run details per managed host, including:

- The first and last time a host was automated.
- The number of times automation has been run or attempted to be run against a host.
- The number of times a managed host has been deleted.

The ability to view the number of times automation has been run on hosts enables you to:

- View your most commonly automated hosts.
- More accurately reflect the scope of your automation landscape.

Soft deletion

Soft deletion enables the removal of decommissioned hosts from the Host Metrics view and excludes them from any potential managed node counting. Additionally, the following host types can also be deleted:

- Ephemeral, uniquely provisioned hosts, such as those used for CI-CD, or testing only.
- Bench provisioning or temporary hosts.
- Older hosts that have been decommissioned and are never automated against.

Soft deletion is intended for legitimate use case scenarios only. It must not be used to violate subscription counting, for example, for the purposes of node recycling. For more information, see [How are "managed nodes" defined as part of the Red Hat Ansible Automation Platform offering?](#)

11.2. Activating Red Hat Ansible Automation Platform

If you are an organization administrator, you must [create a service account](#) and use the client ID and client secret to activate your subscription.

If you do not have administrative access, you can enter your Red Hat username and password in the Client ID and Client secret fields, to locate and add your subscription to your Ansible Automation Platform instance.

NOTE

If you enter your client ID and client secret but cannot locate your subscription, you might not have the correct permissions set on your service account. For more information and troubleshooting guidance for service accounts, see [Configure Ansible Automation Platform to authenticate through service account credentials](#).

For Red Hat Satellite, input your Satellite username and Satellite password in the fields below.

Red Hat Ansible Automation Platform uses available subscriptions or a subscription manifest to authorize the use of Ansible Automation Platform. To obtain a subscription, you can do either of the following:

1. Use your Red Hat username and password, service account credentials, or Satellite credentials when you launch Ansible Automation Platform.

2. Upload a subscriptions manifest file either using the Red Hat Ansible Automation Platform interface or manually in an Ansible Playbook.

11.2.1. Activate with credentials

When Ansible Automation Platform launches for the first time, the Ansible Automation Platform Subscription screen automatically displays. If you are an organization administrator, you can use your Red Hat service account to retrieve and import your subscription directly into Ansible Automation Platform.

If you do not have administrative access, you can enter your Red Hat username and password in the Client ID and Client secret fields, respectively, to locate and add your subscription to your Ansible Automation Platform instance.

NOTE

You are opted in for Automation Analytics by default when you activate the platform on first time log in. This helps Red Hat improve the product by delivering you a much better user experience. You can opt out, after activating Ansible Automation Platform, by doing the following:

1. From the navigation panel, select **Settings › Automation Execution › System**.
2. Clear the **Gather data for Automation Analytics** option.
3. Click **[Save]**.

Procedure

1. Log in to Red Hat Ansible Automation Platform.
2. Select **Service Account / Red Hat Satellite**.
3. Enter your **Client ID / Satellite username** and **Client secret / Satellite password**.
4. Select your subscription from the **Subscription** list.

NOTE

You can also use your Satellite username and password if your cluster nodes are registered to Satellite through Subscription Manager.

5. Review the End User License Agreement and select **I agree to the End User License Agreement**.
6. Click **[Finish]**.

Verification

After your subscription has been accepted, subscription details are displayed. A status of *Compliant* indicates your subscription is in compliance with the number of hosts you have automated within your subscription count. Otherwise, your status will show as *Out of Compliance*, indicating you have exceeded the number of hosts in your subscription. Other important information displayed include the following:

Hosts automated

Host count automated by the job, which consumes the license count

Hosts imported

Host count considering all inventory sources (does not impact hosts remaining)

Hosts remaining

Total host count minus hosts automated

11.2.2. Activate with a manifest file

If you have a subscriptions manifest, you can upload the manifest file either by using the Red Hat Ansible Automation Platform interface.

You are opted in for Automation Analytics by default when you activate the platform on first time log in. This helps Red Hat improve the product by delivering you a much better user experience. However, you can opt out of Automation Analytics after the Ansible Automation Platform is activated. To opt out, select **Settings** › **Automation Execution** › **System** from the navigation panel, uncheck the **Gather data for Automation Analytics** option, and then click **[Save]**.

11.3. Keeping your subscription in compliance

Your subscription has two possible statuses:

- **Compliant:** Indicates that your subscription is appropriate for the number of hosts that you have automated within your subscription count.
- **Out of compliance:** Indicates that you have exceeded the number of hosts in your subscription.

Compliance is computed as follows:

```
managed > manifest_limit    => non-compliant
managed <= manifest_limit   => compliant
```

Where: **managed** is the number of unique managed hosts without deletions, and **manifest_limit** is the number of managed hosts in the subscription manifest.

Other important information displayed are:

- **Hosts automated:** The number of hosts automated by the job, which consumes the license count.
- **Hosts imported:** The number of hosts considering unique host names across all inventory sources. This number does not impact hosts remaining.
- **Hosts remaining:** The number of hosts minus the number of hosts automated.
- **Hosts deleted:** The number of hosts that were deleted, freeing the license capacity.
- **Active hosts previously deleted:** The number of hosts now active that were previously deleted.

For example, if you have a subscription capacity of 10 hosts:

- Starting with 9 hosts, 2 hosts were added and 3 hosts were deleted, you now have 8 hosts


(compliant).

- 3 hosts were automated again, now you have 11 hosts, which puts you over the subscription limit of 10 (non-compliant).
- If you delete hosts, refresh the subscription details to see the change in count and status.

11.3.1. Viewing hosts automated in the user interface

Procedure

1. In the navigation panel, select **Automation Analytics** › **Host Metrics** to view the activity associated with hosts, such as those that have been automated and deleted.

Each unique hostname is listed and sorted by the user's preference. [Host metrics]

NOTE

A scheduled task automatically updates these values on a weekly basis and deletes jobs with hosts that were last automated more than a year ago.

2. Delete unnecessary hosts directly from the Host Metrics view by selecting the required hosts and clicking **[Delete]**.

These are soft-deleted, meaning their records are not removed, but are not being used and thereby not counted towards your subscription.

Additional resources

- [Configuring automation execution/index#controller-keep-subscription-in-compliance](#)[Keeping your subscription in compliance]

11.3.2. Viewing Hosts automated in the CLI

Automation controller provides a way to generate a CSV output of the host metric data and host metric summary through the *Command Line Interface* (CLI). You can also soft delete hosts in bulk through the API.

awx-manage utility

To collect and manage host metric data and related cluster information in Ansible Automation Platform, use the **awx-manage** utility.

Procedure

1. The **awx-manage** utility supports the following options:

```
awx-manage host_metric --csv
```

2. This command produces host metric data, a host metrics summary file, and a cluster info file. To package all the files into a single tar file for distribution and sharing, use:

```
awx-manage host_metric --tarball
```

3. To specify the number of rows (<n>) to output to each file:

```
awx-manage host_metric --tarball --rows_per_file <n>
```

Automation Analytics receives and uses the JSON file.

Additional resources

- [Configuring automation execution/assembly-controller-metrics](#)[Usage reporting with metrics-utility]

11.3.3. Deleting Hosts automated using API endpoint

The API lists only non-deleted records and are sortable by last_automation and used_in_inventories columns.

You can use the host metric API endpoint, `api/v2/host_metric` to soft delete hosts, using:

```
api/v2/host_metric <n> DELETE
```

A monthly scheduled task automatically deletes jobs that uses hosts from the Host Metric table that were last automated more than a year ago.

12. Usage reporting with metrics-utility

The Ansible Automation Platform metrics utility tool (`metrics-utility`) is a command-line utility that is installed on a system containing an instance of automation controller.

When installed and configured, `metrics-utility` gathers billing-related metrics from your system and creates a consumption-based billing report. The `metrics-utility` tool is especially suited for users who have multiple managed hosts and want to use consumption-based billing. After a report is generated, it is deposited in a target location that you specify in the configuration file.

`metrics-utility` collects two types of data from your system: configuration data and reporting data.

The configuration data includes the following information:

- Version information for automation controller and for the operating system
- Subscription information
- The base URL

The reporting data includes the following information:

- Job name and ID
- Host name

- Inventory name
- Organization name
- Project name
- Success or failure information
- Report date and time

To ensure that `metrics-utility` continues to work as configured, clear your report directories of outdated reports regularly.

12.1. Configuring metrics-utility

Configure the metrics-utility to gather and report usage data for your Ansible Automation Platform, both on Red Hat Enterprise Linux and OpenShift Container Platform.

12.1.1. Configuring metrics-utility on Red Hat Enterprise Linux

Prerequisites:

- An active Ansible Automation Platform subscription

Metrics-utility is included with Ansible Automation Platform, so you do not need a separate installation. The following procedure gathers the relevant data and generate a [CCSP](#) report containing your usage metrics. You can configure these commands as cronjobs to ensure they run at the beginning of every month. See [How to schedule jobs using the Linux 'cron' utility](#) for more on configuring using the cron syntax.

Procedure

1. Create two scripts in your user's home directory to set correct variables to ensure that `metrics-utility` gathers all relevant data.
 - a. In `/home/my-user/cron-gather`:

```
#!/bin/sh

# Specify the following variables to indicate where the report is deposited in
your file system
export METRICS_UTILITY_SHIP_TARGET=directory
export METRICS_UTILITY_SHIP_PATH=/awx_devel/awx-dev/metrics-
utility/shipped_data/billing

# Run the following command to gather and store the data in the provided
SHIP_PATH directory:
metrics-utility gather_automation_controller_billing_data --ship --until=10m
```

- b. In `/home/my-user/cron-report`:

```
#!/bin/sh
```

```
# Specify the following variables to indicate where the report is deposited in
your file system
export METRICS_UTILITY_SHIP_TARGET=directory
export METRICS_UTILITY_SHIP_PATH=/awx_devel/awx-dev/metrics-
utility/shipped_data/billing

# Set these variables to generate a report:
export METRICS_UTILITY_REPORT_TYPE=CCSPv2
export METRICS_UTILITY_PRICE_PER_NODE=11.55 # in USD
export METRICS_UTILITY_REPORT_SKU=MCT3752MO
export METRICS_UTILITY_REPORT_SKU_DESCRIPTION="EX: Red Hat Ansible Automation
Platform, Full Support (1 Managed Node, Dedicated, Monthly)"
export METRICS_UTILITY_REPORT_H1_HEADING="CCSP Reporting <Company>: ANSIBLE
Consumption"
export METRICS_UTILITY_REPORT_COMPANY_NAME="Company Name"
export METRICS_UTILITY_REPORT_EMAIL="email@email.com"
export METRICS_UTILITY_REPORT_RHN_LOGIN="test_login"
export METRICS_UTILITY_REPORT_COMPANY_BUSINESS_LEADER="BUSINESS LEADER"
export METRICS_UTILITY_REPORT_COMPANY_PROCUREMENT_LEADER="PROCUREMENT LEADER"

# Build the report
metrics-utility build_report
```

2. To ensure that these files are executable, run:

```
chmod a+x /home/my-user/cron-gather /home/my-user/cron-report
```

3. To open the cron file for editing, run:

```
crontab -e
```

4. To configure the run schedule, add the following parameters to the end of the file and specify how often you want **metrics-utility** to gather information and build a report using **cron syntax**. In the following example, the **gather** command is configured to run every hour at 00 minutes. The **build_report** command is configured to run on the second day of each month at 4:00 AM.

```
0 */1 * * * /home/my-user/cron-gather
0 4 2 * * /home/my-user/cron-report
```

5. Save and close the file.

Verification

Use the following verification steps to ensure correct configuration:

1. To confirm that your **cron** job entries have been saved correctly, run:

```
crontab -l
```

2. Inspect the **cron** log to verify that the **cron** daemon is executing the commands and that **metrics-utility** is producing output:

```
cat /var/log/cron
```

For reference, see the following example output:

```
May  8 09:45:03 ip-10-0-6-23 CROND[51623]: (root) CMDOUT (No billing data for
month: 2024-04)
May  8 09:45:03 ip-10-0-6-23 CROND[51623]: (root) CMDEND (metrics-utility
build_report)
May  8 09:45:19 ip-10-0-6-23 crontab[51619]: (root) END EDIT (root)
May  8 09:45:34 ip-10-0-6-23 crontab[51659]: (root) BEGIN EDIT (root)
May  8 09:46:01 ip-10-0-6-23 CROND[51688]: (root) CMD (metrics-utility
gather_automation_controller_billing_data --ship --until=10m)
May  8 09:46:03 ip-10-0-6-23 CROND[51669]: (root) CMDOUT (/tmp/9e3f86ee-c92e-4b05-
8217-72c496e6ffd9-2024-05-08-093402+0000-2024-05-08-093602+0000-0.tar.gz)
May  8 09:46:03 ip-10-0-6-23 CROND[51669]: (root) CMDEND (metrics-utility
gather_automation_controller_billing_data --ship --until=10m)
May  8 09:46:26 ip-10-0-6-23 crontab[51659]: (root) END EDIT (root)
```

The generated report will have the default name `CCSP-<YEAR>-<MONTH>.xlsx` and is saved in the ship path that you specified in step 1a.

NOTE Time and date might vary depending on how you configure the run schedule.

12.1.2. Configuring `metrics-utility` on OpenShift Container Platform from the Ansible Automation Platform operator

`metrics-utility` is included in the OpenShift Container Platform image beginning with version 4.12, 4.512, and 4.6. If your system does not have `metrics-utility` installed, update your OpenShift image to the latest version.

Complete the following steps to configure the run schedule for `metrics-utility` on OpenShift Container Platform using the Ansible Automation Platform operator:

Creating a ConfigMap in the OpenShift UI YAML view

To inject the `metrics-utility` cronjobs with configuration data, use the following procedure to create a ConfigMap in the OpenShift UI YAML view:

Prerequisites:

- A running OpenShift cluster
- An operator-based installation of Ansible Automation Platform on OpenShift Container Platform.

NOTE `metrics-utility` runs as indicated by the parameters you set in the configuration file. You cannot run the utility manually on OpenShift Container Platform.

Procedure

1. From the navigation panel, select **ConfigMaps**.

2. Click **[Create ConfigMap]**.
3. On the next screen, select the YAML view tab.
4. In the YAML field, enter the following parameters with the appropriate variables set:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: automationcontroller-metrics-utility-configmap
data:
  METRICS_UTILITY_SHIP_TARGET: directory
  METRICS_UTILITY_SHIP_PATH: /metrics-utility
  METRICS_UTILITY_REPORT_TYPE: CCSP
  METRICS_UTILITY_PRICE_PER_NODE: '11' # in USD
  METRICS_UTILITY_REPORT_SKU: MCT3752MO
  METRICS_UTILITY_REPORT_SKU_DESCRIPTION: "EX: Red Hat Ansible Automation Platform,
Full Support (1 Managed Node, Dedicated, Monthly)"
  METRICS_UTILITY_REPORT_H1_HEADING: "CCSP Reporting <Company>: ANSIBLE
Consumption"
  METRICS_UTILITY_REPORT_COMPANY_NAME: "Company Name"
  METRICS_UTILITY_REPORT_EMAIL: "email@email.com"
  METRICS_UTILITY_REPORT_RHN_LOGIN: "test_login"
  METRICS_UTILITY_REPORT_COMPANY_BUSINESS_LEADER: "BUSINESS LEADER"
  METRICS_UTILITY_REPORT_COMPANY_PROCUREMENT_LEADER: "PROCUREMENT LEADER"
```

5. Click **[Create]**.

Verification

- To verify that the ConfigMap was created and **metrics-utility** is installed, select **ConfigMap** from the navigation panel and search for your ConfigMap in the list.

Deploy automation controller

To deploy automation controller and specify variables for how often **metrics-utility** gathers usage information and generates a report, use the following procedure:

Procedure

1. From the navigation panel, select **Installed Operators**.
2. Select **Ansible Automation Platform**.
3. In the Operator details, select the automation controller tab.
4. Click **[Create automation controller]**.
5. Select the YAML view option. The YAML now shows the default parameters for automation controller. The relevant parameters for **metrics-utility** are the following:

Parameter	Variable
metrics-utility-enabled	True.

Parameter	Variable
<code>metrics_utility_cronjob_gather_schedule</code>	@hourly or @daily.
<code>metrics_utility_cronjob_report_schedule</code>	@daily or @monthly.

- Find the `metrics_utility_enabled` parameter and change the variable to true.
- Find the `metrics_utility_cronjob_gather_schedule` parameter and enter a variable for how often the utility should gather usage information (for example, @hourly or @daily).
- Find the `metrics_utility_cronjob_report_schedule` parameter and enter a variable for how often the utility generates a report (for example, @daily or @monthly).
- Click [Create].

12.1.3. Configuring `metrics-utility` on a manual containerized installation of Ansible Automation Platform

`metrics-utility` is included in the OpenShift Container Platform image beginning with version 4.12, 4.512, and 4.6. If your system does not have `metrics-utility` installed, update your OpenShift image to the latest version.

Complete the following steps to configure the run schedule for `metrics-utility` on OpenShift Container Platform using the Ansible Automation Platform operator:

Use the following steps to configure `metrics-utility` on a manual containerized installation of Ansible Automation Platform:

- Prepare host directories and scripts.
- Make scripts executable.
- Configure cron jobs.

Prerequisites

- An active Ansible Automation Platform subscription

Minimum resource requirements

Using the `metrics-utility` tool on a containerized installation of Ansible Automation Platform requires the following resources:

- CPU: 1 dedicated CPU core
 - 100% of 1 core is used during execution
- Memory:
 - Minimum: 256 MB RAM (supports up to ~10,000 job host summaries)
 - Recommended: 512 MB RAM (standard deployments)
 - Large-scale: 1 GB RAM (supports up to ~100,000 job host summaries)

NOTE

Memory requirements scale with the number of hosts and jobs being processed.

- Execution time: Report generation typically completes within 10-30 seconds, depending on data volume

Preparing host directories and scripts

To prepare host directories and scripts, create two shell scripts on your host machine to execute **metrics-utility** commands inside your running container.

Procedure

1. Modify your inventory file to enable metrics-utility container deployment by adding the following line under the [all:vars] section:

```
metrics_utility_enabled=true
```

This setting instructs the installer to create and configure a dedicated **automation-controller-metrics-utility** container as part of your Ansible Automation Platform deployment. Be sure to place this line under the [all:vars] header in your inventory file, alongside your other global variables. Re-run the installer script to activate the container if your Ansible Automation Platform deployment has already been configured.

2. Run the following command to create the host report destination directory:

```
mkdir -p /home/my-user/aap/controller/data/
```

This directory on your host machine stores the final generated reports.

3. Create a file named **/home/my-user/cron-gather** (replace **/home/my-user/** with your desired path) and add the following content to create the **cron-gather** script:

```
#!/bin/sh

# Specify the following variables to indicate where the report is deposited in your
file system
export METRICS_UTILITY_SHIP_TARGET=directory
export METRICS_UTILITY_SHIP_PATH=/metrics_utility/data

# Run metrics-utility inside the running container
podman exec \
  -e METRICS_UTILITY_SHIP_TARGET="$METRICS_UTILITY_SHIP_TARGET" \
  -e METRICS_UTILITY_SHIP_PATH="$METRICS_UTILITY_SHIP_PATH" \
  automation-controller-task \
  metrics-utility gather_automation_controller_billing_data --ship --until=10m
```

This script runs **metrics-utility** to gather billing data from your Ansible Automation Platform instance and store it temporarily inside the container.

4. Create a file named **/home/my-user/cron-report** (replace **/home/my-user/** with your desired path)

and add the following content to create the `cron-report` script:

```
#!/bin/sh

# --- Configuration Variables ---
# Define variables for the metrics utility parameters
export METRICS_UTILITY_SHIP_TARGET="directory"
export METRICS_UTILITY_SHIP_PATH="/metrics_utility/data" # This is the *internal*
path in the container
export METRICS_UTILITY_REPORT_TYPE="CCSPv2"
export METRICS_UTILITY_PRICE_PER_NODE="11.55"
export METRICS_UTILITY_REPORT_SKU="MCT3752MO"
export METRICS_UTILITY_REPORT_SKU_DESCRIPTION="EX: Red Hat {PlatformNameShort},
Full Support (1 Managed Node, Dedicated, Monthly)"
export METRICS_UTILITY_REPORT_H1_HEADING="CCSP Reporting <Company>: ANSIBLE
Consumption"
export METRICS_UTILITY_REPORT_COMPANY_NAME="Company Name"
export METRICS_UTILITY_REPORT_EMAIL="email@email.com"
export METRICS_UTILITY_REPORT_RHN_LOGIN="test_login"
export METRICS_UTILITY_REPORT_COMPANY_BUSINESS_LEADER="BUSINESS LEADER"
export METRICS_UTILITY_REPORT_COMPANY_PROCURMENT_LEADER="PROCUREMENT LEADER"

podman exec \
  -e METRICS_UTILITY_SHIP_TARGET="$METRICS_UTILITY_SHIP_TARGET" \
  -e METRICS_UTILITY_SHIP_PATH="$METRICS_UTILITY_SHIP_PATH" \
  -e METRICS_UTILITY_REPORT_TYPE="$METRICS_UTILITY_REPORT_TYPE" \
  -e METRICS_UTILITY_PRICE_PER_NODE="$METRICS_UTILITY_PRICE_PER_NODE" \
  -e METRICS_UTILITY_REPORT_SKU="$METRICS_UTILITY_REPORT_SKU" \
  -e
METRICS_UTILITY_REPORT_SKU_DESCRIPTION="$METRICS_UTILITY_REPORT_SKU_DESCRIPTION" \
  -e METRICS_UTILITY_REPORT_H1_HEADING="$METRICS_UTILITY_REPORT_H1_HEADING" \
  -e METRICS_UTILITY_REPORT_COMPANY_NAME="$METRICS_UTILITY_REPORT_COMPANY_NAME" \
  -e METRICS_UTILITY_REPORT_EMAIL="$METRICS_UTILITY_REPORT_EMAIL" \
  -e METRICS_UTILITY_REPORT_RHN_LOGIN="$METRICS_UTILITY_REPORT_RHN_LOGIN" \
  -e
METRICS_UTILITY_REPORT_COMPANY_BUSINESS_LEADER="$METRICS_UTILITY_REPORT_COMPANY_BUS
INESS_LEADER" \
  -e
METRICS_UTILITY_REPORT_COMPANY_PROCURMENT_LEADER="$METRICS_UTILITY_REPORT_COMPANY_
PROCUREMENT_LEADER" \
  "automation-controller-metrics-utility" \
  metrics-utility build_report
```

This script `METRICS_UTILITY_REPORT_SKU_DESCRIPTION` `metrics-utility` to build a report from the gathered data and then copy the final report from the container to your specified host directory.

Making the scripts executable

Ensure that both scripts you created are executable by using the following procedure:

Procedure

1. Run the following command:

```
chmod a+x /home/my-user/cron-gather /home/my-user/cron-report
```

NOTE

Ensure that you have replaced `/home/my-user/` with your actual path.

Configuring cron jobs

Configure `cron` to schedule the `metrics-utility` commands to run automatically by using the following procedure:

Procedure

1. Open the `cron` file for editing:

```
crontab -e
```

2. Add the following lines to the end of the `crontab` file to add `cron` schedule entries:

```
0 */1 * * * /home/my-user/cron-gather
0 4 2 * * /home/my-user/cron-report
```

These examples configure the `gather` command to run every hour at 00 minutes and the `build_report` command to run on the second day of each month at 4:00 AM.

3. Adjust the cron syntax as needed to fit your desired schedule. For example:

a. `0 */1 * * *`: Runs at minute 0 of every hour.

b. `0 4 2 * *`: Runs at 04:00 on day-of-month 2.

4. Save and close the file.

Verification

Use the following verification steps to ensure correct configuration: . To confirm that your `cron` job entries have been saved correctly, run:

```
crontab -l
```

If your `cron` job entries have been saved correctly, you will see the two lines you added for `cron-gather` and `cron-report`. . Inspect the `cron` log to check if the `cron` daemon is executing the commands and if `metrics-utility` is producing output:

```
cat /var/log/cron
```

For reference, see the following example output:

Example Output:

```
Aug 20 09:55:01 aap CROND[72746]: (aap) CMD (/home/aap/Documents/scripts/cron-gather)
Aug 20 09:55:03 aap CROND[72744]: (aap) CMDOUT (2025-08-20 08:55:03,581 INFO      [-]
awx.main.analytics /tmp/3292ca44-3314-4f0b-b3f6-ba4a1e47a2b1-2025-08-20-083506+0000-
2025-08-20-084503+0000-0.tar.gz)
Aug 20 09:55:03 aap CROND[72744]: (aap) CMDOUT (/tmp/3292ca44-3314-4f0b-b3f6-
ba4a1e47a2b1-2025-08-20-083506+0000-2025-08-20-084503+0000-0.tar.gz)
Aug 20 09:55:03 aap CROND[72744]: (aap) CMDOUT (2025-08-20 08:55:03,581 INFO      [-]
awx.main.analytics /tmp/3292ca44-3314-4f0b-b3f6-ba4a1e47a2b1-2025-08-20-083506+0000-
2025-08-20-084503+0000-1.tar.gz)
Aug 20 09:55:03 aap CROND[72744]: (aap) CMDOUT (/tmp/3292ca44-3314-4f0b-b3f6-
ba4a1e47a2b1-2025-08-20-083506+0000-2025-08-20-084503+0000-1.tar.gz)
Aug 20 09:55:03 aap CROND[72744]: (aap) CMDEND (/home/aap/Documents/scripts/cron-
gather)
Aug 20 10:00:02 aap CROND[77629]: (aap) CMD (/home/aap/Documents/scripts/cron-report)
Aug 20 10:00:02 aap CROND[77623]: (aap) CMDOUT (Generating metrics report inside the
container...)
Aug 20 10:00:04 aap CROND[77623]: (aap) CMDOUT (2025-08-20 09:00:04,639 INFO      [-]
awx.main.analytics No billing data for month 2025-07)
Aug 20 10:00:04 aap CROND[77623]: (aap) CMDOUT (No billing data for month 2025-07)
Aug 20 10:00:05 aap CROND[77623]: (aap) CMDOUT (Copying generated reports from
container to host: /tmp/ansible_metrics_reports)
Aug 20 10:00:05 aap CROND[77623]: (aap) CMDOUT (Reports successfully moved to:
/tmp/ansible_metrics_reports)
Aug 20 10:00:05 aap CROND[77623]: (aap) CMDOUT (Files in destination:)
Aug 20 10:00:05 aap CROND[77623]: (aap) CMDOUT (total 0)
Aug 20 10:00:05 aap CROND[77623]: (aap) CMDOUT (drwxr-xr-x. 3 aap aap 18 Aug 20 09:43
data)
Aug 20 10:00:05 aap CROND[77623]: (aap) CMDEND (/home/aap/Documents/scripts/cron-
report)
```

1. Locate generated reports: The generated report has a default name format of **CCSP-<YEAR>-<MONTH>.xlsx** (for example, **CCSP-2024-07.xlsx**). The report is deposited in the **METRICS_UTILITY_SHIP_PATH** with the prefixed path you specified in Step 2 (for example, **/home/my-user/aap/controller/data/metrics_utility/data**).

12.2. Fetching a monthly report

Fetch a monthly report from Ansible Automation Platform to gather usage metrics and create a consumption-based billing report. To fetch a monthly report on Red Hat Enterprise Linux or on OpenShift Container Platform, use the following procedures:

12.2.1. Fetching a monthly report on Red Hat Enterprise Linux

Use the following procedure to fetch a monthly report on Red Hat Enterprise Linux:

Procedure

- Run: `scp -r username@controller_host:$METRICS_UTILITY_SHIP_PATH/data/<YYYY>/<MM>/ /local/directory/`

The system saves the generated report as `CCSP-<YEAR>-<MONTH>.xlsx` in the ship path that you specified.

12.2.2. Fetching a monthly report on OpenShift Container Platform from the Ansible Automation Platform Operator

Procedure

1. Use the following playbook to fetch a monthly consumption report for Ansible Automation Platform on OpenShift Container Platform:

NOTE

To use this playbook, you must have the `kubernetes.core.k8s` collection to be installed on your machine.

```
# Requires Ansible and Kubernetes.core collection
- name: Copy directory from Kubernetes PVC to local machine
  hosts: "{{ host | default(omit) }}"

vars:
  report_dir_path: "/mnt/metrics/reports/{{ year }}/{{ month }}/"
  data_files_dir_path: "/mnt/metrics/data/{{ year }}/{{ month }}/{{ day }}"

tasks:
  - name: Create a temporary pod to access PVC data
    kubernetes.core.k8s:
      definition:
        apiVersion: v1
        kind: Pod
        metadata:
          name: temp-pod
          namespace: "{{ namespace_name }}"
      spec:
        containers:
          - name: busybox
            image: busybox
            command: ["/bin/sh"]
            args: ["-c", "sleep 3600"] # Keeps the container alive for 1 hour
            volumeMounts:
              - name: "{{ pvc }}"
                mountPath: "/mnt/metrics"
        volumes:
          - name: "{{ pvc }}"
```

```

        persistentVolumeClaim:
            claimName: automationcontroller-metrics-utility
            restartPolicy: Never
        register: pod_creation

- name: Wait for both initContainer and main container to be ready
  kubernetes.core.k8s_info:
    kind: Pod
    namespace: "{{ namespace_name }}"
    name: temp-pod
  register: pod_status
  until: >
    pod_status.resources[0].status.containerStatuses[0].ready
  retries: 30
  delay: 10

- name: Create a tarball of the directory of the report in the container
  kubernetes.core.k8s_exec:
    namespace: "{{ namespace_name }}"
    pod: temp-pod
    container: busybox
    command: tar czf /tmp/metrics.tar.gz -C "{{ report_dir_path }}" .
  register: tarball_creation

- name: Create a tarball of the directory of the data files in the container
  kubernetes.core.k8s_exec:
    namespace: "{{ namespace_name }}"
    pod: temp-pod
    container: busybox
    command: tar czf /tmp/data_files.tar.gz -C "{{ data_files_dir_path }}" .
  register: tarball_creation_files

- name: Copy the report tarball from the container to the local machine
  kubernetes.core.k8s_cp:
    namespace: "{{ namespace_name }}"
    pod: temp-pod
    container: busybox
    state: from_pod
    remote_path: /tmp/metrics.tar.gz
    local_path: "{{ local_dir }}/metrics.tar.gz"
  when: tarball_creation is succeeded

- name: Copy the data files tarball from the container to the local machine
  kubernetes.core.k8s_cp:
    namespace: "{{ namespace_name }}"
    pod: temp-pod
    container: busybox
    state: from_pod
    remote_path: /tmp/data_files.tar.gz
    local_path: "{{ local_dir }}/data_files.tar.gz"
  when: tarball_creation_files is succeeded

```

```

- name: Ensure the local directory exists
  ansible.builtin.file:
    path: "{{ local_dir }}"
    state: directory
    mode: '0755'

- name: Extract the report tarball on the local machine
  ansible.builtin.unarchive:
    src: "{{ local_dir }}/metrics.tar.gz"
    dest: "{{ local_dir }}"
    remote_src: true
    extra_opts: "--strip-components=1"
  when: tarball_creation is succeeded

- name: Extract the data files tarball on the local machine
  ansible.builtin.unarchive:
    src: "{{ local_dir }}/data_files.tar.gz"
    dest: "{{ local_dir }}"
    remote_src: true
    extra_opts: "--strip-components=1"
    list_files: true
  register: unarchive_result
  when: tarball_creation_files is succeeded

- name: Extract the extracted data files tarball on the local machine
  ansible.builtin.unarchive:
    src: "{{ local_dir }}/{{ item }}"
    dest: "{{ local_dir }}"
    remote_src: true
    extra_opts: "--strip-components=1"
  loop: "{{ unarchive_result.files }}"
  when: tarball_creation_files is succeeded
  ignore_errors: true # noqa ignore-errors

- name: Delete the temporary pod
  kubernetes.core.k8s:
    api_version: v1
    kind: Pod
    namespace: "{{ namespace_name }}"
    name: temp-pod
    state: absent

```

12.3. Modifying the run schedule

You can configure `metrics-utility` to run at specified times and intervals. Run frequency is expressed in cronjobs. For more information on the cron utility, see [How to schedule jobs using the Linux ‘Cron’ utility](#).

To modify the run schedule on Red Hat Enterprise Linux and on OpenShift Container Platform, use

one of the following procedures:

Procedure

1. From the command line, run:

```
crontab -e
```

2. After the code editor has opened, update the `gather` and `build` parameters using cron syntax as shown below:

```
*/2 * * * * metrics-utility gather_automation_controller_billing_data --ship --until=10m
```

```
*/* * * * * metrics-utility build_report
```

3. Save and close the file.

12.3.1. Modifying the run schedule on OpenShift Container Platform from the Ansible Automation Platform operator

To adjust the execution schedule of the `metrics-utility` within your Ansible Automation Platform deployment running on OpenShift Container Platform, use the following procedure:

Procedure

1. From the navigation panel, select **Workloads > Deployments**.
2. On the next screen, select **automation-controller-operator-controller-manager**.
3. Beneath the heading **Deployment Details**, click the down arrow button to change the number of pods to zero. This pauses the deployment so you can update the running schedule.
4. From the navigation panel, select **Installed Operators**.
5. From the list of installed operators, select Ansible Automation Platform.
6. On the next screen, select the automation controller tab.
7. From the list that appears, select your automation controller instance.
8. On the next screen, select the **YAML** tab.
9. In the **YAML** file, find the following parameters and enter a variable representing how often `metrics-utility` should gather data and how often it should produce a report:

`metrics_utility_cronjob_gather_schedule:`
`metrics_utility_cronjob_report_schedule:`
10. Click **[Save]**.
11. From the navigation menu, select **Deployments** and then select **automation-controller-operator-controller-manager**.
12. Increase the number of pods to 1.
13. To verify that you have changed the `metrics-utility` running schedule successfully, you can take one or both of the following steps:
 - a. Return to the **YAML** file and ensure that the previously described parameters reflect the

correct variables.

- b. From the navigation menu, select **Workloads** › **Cronjobs** and ensure that your cronjobs show the updated schedule.

12.4. Supported storage

Supported storage is available for storing the raw data obtained by using the `metrics-utility gather_automation_controller_billing_data` command and storing the generated reports obtained by using the `metrics-utility build_report` command. Apply the environment variables to this storage based on your Ansible Automation Platform installation.

12.4.1. Local disk

For an installation of Ansible Automation Platform on Red Hat Enterprise Linux, the default storage option is a local disk. Using an OpenShift deployment of OpenShift Container Platform, default storage is a path inside the attached Persistent Volume Claim.

+

```
# Set needed ENV VARs for gathering data and generating reports
export METRICS_UTILITY_SHIP_TARGET=directory
# Your path on the local disk
export METRICS_UTILITY_SHIP_PATH=/path_to_data_and_reports/...
```

12.4.2. Object storage with S3 interface

To use object storage with S3 interface, for example, with AWS S3, Ceph Object storage, or Minio, you must define environment variables for data gathering and report building commands and cronjobs.

+

```
#####
export METRICS_UTILITY_SHIP_TARGET=s3
# Your path in the object storage
export METRICS_UTILITY_SHIP_PATH=path_to_data_and_reports/...

#####
# Define S3 config
export METRICS_UTILITY_BUCKET_NAME=metricsutilities3
export METRICS_UTILITY_BUCKET_ENDPOINT="https://s3.us-east-1.amazonaws.com"
# For AWS S3, define also a region
export METRICS_UTILITY_BUCKET_REGION="us-east-1"

#####
# Define S3 credentials
export METRICS_UTILITY_BUCKET_ACCESS_KEY=<access_key>
```

```
export METRICS_UTILITY_BUCKET_SECRET_KEY=<secret_key>
```

12.5. Report types

This section provides additional configurations for data gathering and report building based on a report type. Apply the environment variables to each report type based on your Ansible Automation Platform installation.

12.5.1. CCSPv2

CCSPv2 is a report which shows the following:

- Directly and indirectly managed node usage
- The content of all inventories
- Content usage

The primary use of this report is for partners under the [CCSP](#) program, but all customers can use it to obtain on-premise reporting showing managed nodes, jobs and content usage across their automation controller organizations.

Set the report type using `METRICS_UTILITY_REPORT_TYPE=CCSPv2`.

12.5.2. Optional collectors for `gather` command

You can use the following optional collectors for the `gather` command:

- `main_jobhostsummary`
 - If present by default, this incrementally collects data from the `main_jobhostsummary` table in the automation controller database, containing information about jobs runs and managed nodes automated.
- `main_host`
 - This collects daily snapshots of the `main_host` table in the automation controller database and has managed nodes and hosts present across automation controller inventories.
- `main_jobevent`
 - This incrementally collects data from the `main_jobevent` table in the automation controller database and contains information about which modules, roles, and Ansible collections are being used.
- `main_indirectmanagednodeaudit`
 - This incrementally collects data from the `main_indirectmanagednodeaudit` table in the automation controller database and contains information about indirectly managed nodes.

```
# Example with all optional collectors
export
METRICS_UTILITY_OPTIONAL_COLLECTORS="main_host,main_jobevent,main_indirectmanage
```

12.5.3. Optional sheets for **build_report** command

You can use the following optional sheets for the **build_report** command:

- **ccsp_summary**
 - This is a landing page specifically for partners under CCSP program. This report takes additional parameters to customize the summary page. For more information, see the following example:

```
export METRICS_UTILITY_PRICE_PER_NODE=11.55 # in USD
export METRICS_UTILITY_REPORT_SKU=MCT3752MO
export METRICS_UTILITY_REPORT_SKU_DESCRIPTION="EX: Red Hat Ansible Automation
Platform, Full Support (1 Managed Node, Dedicated, Monthly)"
export METRICS_UTILITY_REPORT_H1_HEADING="CCSP NA Direct Reporting Template"
export METRICS_UTILITY_REPORT_COMPANY_NAME="Partner A"
export METRICS_UTILITY_REPORT_EMAIL="email@email.com"
export METRICS_UTILITY_REPORT_RHN_LOGIN="test_login"
export METRICS_UTILITY_REPORT_PO_NUMBER="123"
export METRICS_UTILITY_REPORT_END_USER_COMPANY_NAME="Customer A"
export METRICS_UTILITY_REPORT_END_USER_CITY="Springfield"
export METRICS_UTILITY_REPORT_END_USER_STATE="TX"
export METRICS_UTILITY_REPORT_END_USER_COUNTRY="US"
```

- **jobs**
 - This is a list of automation controller jobs launched. It is grouped by job template.
- **managed_nodes**
 - This is a deduplicated list of managed nodes automated by automation controller.
- **indirectly_managed_nodes**
 - This is a deduplicated list of indirect managed nodes automated by automation controller.
- **infrastructure_summary**
 - This additional tab summarizes the infrastructure taxonomy for indirect nodes in three levels:
 1. Infrastructure
 2. Device category
 3. Device type

Taxonomy mapping is dependent on the facts column in the raw data.

- **inventory_scope**
 - This is a deduplicated list of managed nodes present across all inventories of automation

controller.

- `usage_by_organizations`
 - This is a list of all automation controller organizations with several metrics showing the organizations usage. This provides data suitable for doing internal chargeback.
- `usage_by_collections`
 - This is a list of Ansible collections used in a automation controller job runs.
- `usage_by_roles`
 - This is a list of roles used in automation controller job runs.
- `usage_by_modules`
 - This is a list of modules used in automation controller job runs.
- `managed_nodes_by_organization`
 - This generates a sheet per organization, listing managed nodes for every organization with the same content as the `managed_nodes` sheet.
- `data_collection_status`
 - This generates a sheet with the status of every data collection done by the `gather` command for the date range the report is built for.

To outline the quality of data collected it also lists:

- unusual gaps between collections (based on `collection_start_timestamp`)
- gaps in collected intervals (based on `since` vs `until`)

```
# Example with all optional sheets
export
METRICS_UTILITY_OPTIONAL_CCSP_REPORT_SHEETS='ccsp_summary,jobs,managed_nodes,indirectly_managed_nodes,inventory_scope,usage_by_organizations,usage_by_collections,usage_by_roles,usage_by_modules,data_collection_status'
```

12.5.4. Filtering reports by organization

To filter your report so that only certain organizations are present, use this environment variable with a semicolon separated list of organization names.

```
export METRICS_UTILITY_ORGANIZATION_FILTER="ACME;Organization 1"
```

This renders only the data from these organizations in the built report. This filter currently does not have any effect on the following optional sheets:

- `usage_by_collections`
- `usage_by_roles`
- `usage_by_modules`

12.5.5. Selecting a date range for your CCSPv2 report

The default behavior of the CCSPv2 report is to build a report for the previous month. The following examples describe how to override this default behavior to select a specific date range for your report:

```
# Build report for a specific month
metrics-utility build_report --month=2025-03

# Build report for a specific date range, including the provided days
metrics-utility build_report --since=2025-03-01 --until=2025-03-31

# Build report for a last 6 months from a current date
metrics-utility build_report --since=6months

# Build report for a last 6 months from a current date overriding an existing report
metrics-utility build_report --since=6months --force
```

12.5.6. RENEWAL_GUIDANCE

The **RENEWAL_GUIDANCE** report provides historical usage from the HostMetric table, applying deduplication and showing real historical usage for renewal guidance purposes.

To generate this report, set the report type to **METRICS_UTILITY_REPORT_TYPE=RENEWAL_GUIDANCE**.

IMPORTANT

This report is currently a tech preview solution. It is designed to provide more information than automation controller when built in the **awx-manage host_metric** command.

Storage and invocation

The **RENEWAL_GUIDANCE** report supports the use of only local disk storage to store the report results. This report does not have a gather data step. It reads directly from the controller HostMetric table, so it does not store any raw data under the **METRICS_UTILITY_SHIP_PATH**.

```
# All parameters the RENEWAL_GUIDANCE report needs
export METRICS_UTILITY_SHIP_TARGET=controller_db
export METRICS_UTILITY_REPORT_TYPE=RENEWAL_GUIDANCE
export METRICS_UTILITY_SHIP_PATH=/path_to_built_report/...

# Will generate report for 12 months back with ephemeral nodes being nodes
# automated for less than 1 month.
metrics-utility build_report --since=12months --ephemeral=1month
```

Showing ephemeral usage

The **RENEWAL_GUIDANCE** report has the capability to list additional sheets with ephemeral usage if the

`ephemeral` parameter is provided. Using the parameter `--ephemeral=1month`, you can define ephemeral nodes as any managed node that has been automated for a maximum of one month, then never automated again. Using this parameter, the total ephemeral usage of the 12-month period is computed as maximum ephemeral nodes used over all 1-month rolling date windows. This sheet is also added into the report.

```
# Will generate report for 12 months back with epehemeral nodes being nodes
# automated for less than 1 month.
metrics-utility build_report --since=12months --ephemeral=1month
```

Selecting a date range for your `RENEWAL_GUIDANCE` report

The `RENEWAL_GUIDANCE` report requires a `since` parameter as the parameter is not supported due to the nature of the HostMetric data and is always set to `now`. To override a report date range that has already been built, use parameter `force` with the command. For more information, see the following examples:

```
# Build report for a specific date range, including the provided days
metrics-utility build_report --since=2025-03-01

# Build report for a last 12 months from a current date
metrics-utility build_report --since=12months

# Build report for a last 12 months from a current date overriding an existing report
metrics-utility build_report --since=12months --force
```

12.5.7. CCSP

`CCSP` is the original report format. It does not include many of the customization of `CCSPv2`, and it is intended to be used only for the CCSP partner program.

12.5.8. Optional collectors for `gather` command

You can use the following optional collectors for the `gather` command:

- `main_jobhostsummary`
 - If present by default, this incrementally collects the `main_jobhostsummary` table from the automation controller database, containing information about jobs runs and managed nodes automated.
- `main_host`
 - This collects daily snapshots of the `main_host` table from the automation controller database and has managed nodes/hosts present across automation controller inventories,
- `main_jobevent`
 - This incrementally collects the `main_jobevent` table from the automation controller database and contains information about which modules, roles, and ansible collections are being

used.

- `main_indirectmanagednodeaudit`
 - This incrementally collects the `main_indirectmanagednodeaudit` table from the automation controller database and contains information about indirectly managed nodes,

```
# Example with all optional collectors
export
METRICS_UTILITY_OPTIONAL_COLLECTORS="main_host,main_jobevent,main_indirectmanagednodeaudit"
```

12.5.9. Optional sheets for `build_report` command

You may use the following optional sheets for the `build_report` command:

- `ccsp_summary`
 - This is a landing page specifically for partners under the CCSP program. It shows managed node usage by each automation controller organization.
 - This report takes additional parameters to customize the summary page. For more information, see the following example:

```
export METRICS_UTILITY_PRICE_PER_NODE=11.55 # in USD
export METRICS_UTILITY_REPORT_SKU=MCT3752MO
export METRICS_UTILITY_REPORT_SKU_DESCRIPTION="EX: Red Hat Ansible Automation Platform, Full Support (1 Managed Node, Dedicated, Monthly)"
export METRICS_UTILITY_REPORT_H1_HEADING="CCSP Reporting <Company>: ANSIBLE Consumption"
export METRICS_UTILITY_REPORT_COMPANY_NAME="Company Name"
export METRICS_UTILITY_REPORT_EMAIL="email@email.com"
export METRICS_UTILITY_REPORT_RHN_LOGIN="test_login"
export METRICS_UTILITY_REPORT_COMPANY_BUSINESS_LEADER="BUSINESS LEADER"
export METRICS_UTILITY_REPORT_COMPANY_PROCUREMENT_LEADER="PROCUREMENT LEADER"
```

- `managed_nodes`
 - This is a deduplicated list of managed nodes automated by automation controller.
- `indirectly_managed_nodes`
 - This is a deduplicated list of indirect managed nodes automated by automation controller.
- `inventory_scope`
 - This is a deduplicated list of managed nodes present across all inventories of automation controller.
- `usage_by_collections`
 - This is a list of Ansible collections used in automation controller job runs.
- `usage_by_roles`

- This is a list of roles used in automation controller job runs. `*usage_by_modules`
- This is a list of modules used in automation controller job runs.

```
# Example with all optional sheets
export
METRICS_UTILITY_OPTIONAL_CCSP_REPORT_SHEETS='ccsp_summary,managed_nodes,indirectly_managed_nodes,inventory_scope,usage_by_collections,usage_by_roles,usage_by_modules'
```

12.5.10. Selecting a date range for your CCSP report

The default behavior of this report is to build a report for the previous month. The following examples describe how to override this default behavior to select a specific date range for your report:

```
# Builds report for a previous month
metrics-utility build_report

# Build report for a specific month
metrics-utility build_report --month=2025-03

# Build report for a specific month overriding an existing report
metrics-utility build_report --month=2025-03 --force
```

12.6. Deduplication of hosts for `metrics-utility` reports

Deduplication changes how `metrics-utility` merges individual host records into countable managed nodes when building reports. Deduplication identifies identical hosts to ensure an accurate count of unique hosts.

`metrics-utility` tracks individual hosts based on their hostnames. Any entries that use the same hostname are tracked as the same host. Additional deduplication strategies are also available using the following environment variable: `METRICS_UTILITY_DEDUPLICATOR=`.

12.6.1. Deduplication in the `RENEWAL_GUIDANCE` report

The default value for `METRICS_UTILITY_DEDUPLICATOR=renewal`. This is the original method, which analyzes `host_name`, `ansible_host`, `ansible_product_serial`, and `ansible_machine_id` separately, and merges entries when any of these items are duplicated.

`METRICS_UTILITY_DEDUPLICATOR=renewal` applies deduplication in multiple iterations. It is limited by the `REPORT_RENEWAL_GUIDANCE_DEDUP_ITERATIONS` environment variable, which defaults to 3.

You can also run `METRICS_UTILITY_DEDUPLICATOR` with the following environment variables:

- `METRICS_UTILITY_DEDUPLICATOR=renewal-hostname`. This is similar to `ccsp`, again preferring `ansible_host` over `host_name` when present. No other fields are considered.

- `METRICS_UTILITY_DEDUPLICATOR=renewal-experimental`. This is similar to `ccsp-experimental`, which first applies the hostname-based deduplication, then deduplicates again, merging when both of the serials match.

12.6.2. Deduplication in the CCSP or CCSPv2 reports

The default value for `METRICS_UTILITY_DEDUPLICATOR= ccsp`. This limits deduplication to hostnames only.

The `ansible_host` variable, from `main_host.variables`, is preferred over `host_name`, from `main_jobhostsummary`, when present.

You can also set `METRICS_UTILITY_DEDUPLICATOR=ccsp-experimental`. This setting merges entries when both their `ansible_product_serial` and `ansible_machine_id` facts are present and duplicated.

13. Secret management system

Users and system administrators upload machine and cloud credentials so that automation can access machines and external services on their behalf. By default, sensitive credential values such as SSH passwords, SSH private keys, and API tokens for cloud services are stored in the database after being encrypted.

With external credentials backed by credential plugins, you can map credential fields (such as a password or an SSH Private key) to values stored in a `secret management system` instead of providing them to automation controller directly.

Automation controller provides a secret management system that include integrations for:

- AWS Secrets Manager Lookup
- Centrify Vault Credential Provider Lookup
- *CyberArk Central Credential Provider* Lookup (CCP)
- CyberArk Conjur Secrets Manager Lookup
- HashiCorp Vault *Key-Value* Store (KV)
- HashiCorp Vault SSH Secrets Engine
- Microsoft Azure *Key Management System* (KMS)
- Thycotic DevOps Secrets Vault
- Thycotic Secret Server
- GitHub app token lookup

These external secret values are fetched before running a playbook that needs them.

13.1. Additional resources

- [Managing user credentials](#).

13.2. Configuring and linking secret lookups

When pulling a secret from a third-party system, you are linking credential fields to external systems. To link a credential field to a value stored in an external system, select the external credential corresponding to that system and provide **metadata** to look up the required value. The metadata input fields are part of the external credential type definition of the source credential.


Automation controller provides a credential plugin interface for developers, integrators, system administrators, and power-users with the ability to add new external credential types to extend it to support other secret management systems.

Use the following procedure to use automation controller to configure and use each of the supported third-party secret management systems.

Procedure

1. Create an external credential for authenticating with the secret management system. At minimum, give a name for the external credential and select one of the following for the **Credential type** field:
 - [AWS Secrets Manager Lookup](#)
 - [Centrify Vault Credential Provider Lookup](#)
 - [CyberArk Central Credential Provider \(CCP\) Lookup](#)
 - [CyberArk Conjur Secrets Manager Lookup](#)
 - [HashiCorp Vault Secret Lookup](#)
 - [HashiCorp Vault Signed SSH](#)
 - [Microsoft Azure Key Vault](#)
 - [Thycotic DevOps Secrets Vault](#)
 - [Thycotic Secret Server](#)
 - [Configuring a GitHub App Installation Access Token Lookup](#)

In this example, the *Demo Credential* is the target credential.

2. For any of the fields that follow the **Type Details** area that you want to link to the external credential, click the key  icon in the input field to link one or more input fields to the external credential along with metadata for locating the secret in the external system.
3. Select the input source to use to retrieve your secret information.
4. Select the credential you want to link to, and click **[Next]**. This takes you to the **Metadata** tab of the input source. This example shows the Metadata prompt for HashiVault Secret Lookup. Metadata is specific to the input source you select.

For more information, see the [Metadata for credential input sources](#) table.

5. Click **[Test]** to verify connection to the secret management system. If the lookup is unsuccessful, an error message displays:
6. Click **[OK]**. You return to the **Details** screen of your target credential.

7. Repeat these steps, starting with Step 3 to complete the remaining input fields for the target credential. By linking the information in this manner, automation controller retrieves sensitive information, such as username, password, keys, certificates, and tokens from the third-party management systems and populates the remaining fields of the target credential form with that data.
8. If necessary, supply any information manually for those fields that do not use linking as a way of retrieving sensitive information. For more information about each of the fields, see the appropriate [Credential types](#).
9. Click **[Save]**.

Additional resources

For more information, see the development documents for [Credential plugins](#).

13.2.1. Metadata for credential input sources

The information required for the **Metadata** tab of the input source.

AWS Secrets Manager Lookup

Metadata	Description
AWS Secrets Manager Region (required)	The region where the secrets manager is located.
AWS Secret Name (required)	Specify the AWS secret name that was generated by the AWS access key.

Centrify Vault Credential Provider Lookup

Metadata	Description
Account name (required)	Name of the system account or domain associated with Centrify Vault.
System Name	Specify the name used by the Centrify portal.

CyberArk Central Credential Provider Lookup

Metadata	Description
Object Query (Required)	Lookup query for the object.
Object Query Format	Select Exact for a specific secret name, or Regex for a secret that has a dynamically generated name.
Object Property	Specifies the name of the property to return. For example, UserName or Address other than the default of Content .
Reason	If required for the object's policy, supply a reason for checking out the secret, as CyberArk logs those.

CyberArk Conjur Secrets Lookup

Metadata	Description
Secret Identifier	The identifier for the secret.
Secret Version	Specify a version of the secret, if necessary, otherwise, leave it empty to use the latest version.

HashiVault Secret Lookup

Metadata	Description
Name of Secret Backend	Specify the name of the KV backend to use. Leave it blank to use the first path segment of the Path to Secret field instead.
Path to Secret (required)	Specify the path to where the secret information is stored; for example, <code>/path/username</code> .
Key Name (required)	Specify the name of the key to look up the secret information.
Secret Version (V2 Only)	Specify a version if necessary, otherwise, leave it empty to use the latest version.

HashiCorp Signed SSH

Metadata	Description
Unsigned Public Key (required)	Specify the public key of the certificate you want to have signed. It needs to be present in the authorized keys file of the target hosts.
Path to Secret (required)	Specify the path to where the secret information is stored; for example, <code>/path/username</code> .
Role Name (required)	A role is a collection of SSH settings and parameters that are stored in Hashi vault. Typically, you can specify some with different privileges or timeouts, for example. So you could have a role that is permitted to get a certificate signed for root, and other less privileged ones, for example.
Valid Principals	Specify a user (or users) other than the default, that you are requesting vault to authorize the cert for the stored key. Hashi vault has a default user for whom it signs, for example, <code>ec2-user</code> .

Microsoft Azure KMS

Metadata	Description
Secret Name (required)	The name of the secret as it is referenced in Microsoft Azure's Key vault app.
Secret Version	Specify a version of the secret, if necessary, otherwise, leave it empty to use the latest version.

Thycotic DevOps Secrets Vault

Metadata	Description
Secret Path (required)	Specify the path to where the secret information is stored, for example, /path/username.

Thycotic Secret Server

Metadata	Description
Secret ID (required)	The identifier for the secret.
Secret Field	Specify the field to be used from the secret.

13.2.2. AWS Secrets Manager lookup

This plugin enables Amazon Web Services to be used as a credential input source to pull secrets from the Amazon Web Services Secrets Manager. The AWS Secrets Manager provides similar service to Microsoft Azure Key Vault, and the AWS collection provides a lookup plugin for it.

When AWS Secrets Manager lookup is selected for **Credential type**, give the following metadata to configure your lookup:

- **AWS Access Key** (required): give the access key used for communicating with AWS key management system
- **AWS Secret Key** (required): give the secret as obtained by the AWS IAM console

13.2.3. Centrify Vault Credential Provider Lookup

You need the Centrify Vault web service running to store secrets for this integration to work. When you select **Centrify Vault Credential Provider Lookup** for **Credential Type**, give the following metadata to configure your lookup:

- **Centrify Tenant URL** (required): give the URL used for communicating with Centrify's secret management system
- **Centrify API User** (required): give the username
- **Centrify API Password** (required): give the password
- **OAuth2 Application ID** : specify the identifier given associated with the OAuth2 client
- **OAuth2 Scope** : specify the scope of the OAuth2 client

13.2.4. CyberArk Central Credential Provider (CCP) Lookup

The CyberArk Central Credential Provider web service must be running to store secrets for this integration to work. When you select **CyberArk Central Credential Provider Lookup** for **Credential Type**, give the following metadata to configure your lookup:

- **CyberArk CCP URL** (required): give the URL used for communicating with CyberArk CCP's secret management system. It must include the URL scheme such as http or https.
- Optional: **Web Service ID**: specify the identifier for the web service. Leaving this blank defaults

to AIMWebService.

- **Application ID** (required): specify the identifier given by CyberArk CCP services.
- **Client Key**: paste the client key if provided by CyberArk.
- **Client Certificate**: include the **BEGIN CERTIFICATE** and **END CERTIFICATE** lines when pasting the certificate, if provided by CyberArk.
- **Verify SSL Certificates**: this option is only available when the URL uses HTTPS. Check this option to verify that the server's SSL/TLS certificate is valid and trusted. For environments that use internal or private CA's, leave this option unchecked to disable verification.

13.2.5. CyberArk Conjurer Secrets Manager Lookup

With a Conjurer Cloud tenant available to target, configure the CyberArk Conjurer Secrets Lookup external management system credential plugin.

When you select **CyberArk Conjurer Secrets Manager Lookup** for **Credential Type**, give the following metadata to configure your lookup:

- **Conjur URL** (required): provide the URL used for communicating with CyberArk Conjurer's secret management system. This must include the URL scheme, such as http or https.
- **API Key** (required): provide the key given by your Conjurer admin
- **Account** (required): the organization's account name
- **Username** (required): the specific authenticated user for this service
- **Public Key Certificate**: include the **BEGIN CERTIFICATE** and **END CERTIFICATE** lines when pasting the public key, if provided by CyberArk

13.2.6. HashiCorp Vault Secret Lookup

When you select **HashiCorp Vault Secret Lookup** for **Credential Type**, give the following metadata to configure your lookup:

- **Server URL** (required): give the URL used for communicating with HashiCorp Vault's secret management system.
- **Token**: specify the access token used to authenticate HashiCorp's server.
- **CA Certificate**: specify the CA certificate used to verify HashiCorp's server.
- **AppRole role_id**: specify the ID if using AppRole for authentication.
- **AppRole secret_id**: specify the corresponding secret ID for AppRole authentication.
- **Client Certificate**: specify a PEM-encoded client certificate when using the TLS authentication method, including any required intermediate certificates expected by Hashicorp Vault.
- **Client Certificate Key**: specify a PEM-encoded certificate private key when using the TLS authentication method.
- **TLS Authentication Role**: specify the role or certificate name in Hashicorp Vault that corresponds to your client certificate when using the TLS authentication method. If it is not provided, Hashicorp Vault attempts to match the certificate automatically.

- **Namespace name:** specify the Namespace name (Hashicorp Vault enterprise only).
- **Kubernetes role:** specify the role name when using Kubernetes authentication.
- **Username:** enter the username of the user to be used to authenticate this service.
- **Password:** enter the password associated with the user to be used to authenticate this service.
- **Path to Auth:** specify a path if other than the default path of `/approle`.
- **API Version** (required): select v1 for static lookups and v2 for versioned lookups.

LDAP authentication requires LDAP to be configured in HashiCorp's Vault UI and a policy added to the user. Cubbyhole is the name of the default secret mount. If you have proper permissions, you can create other mounts and write key values to those.

To test the lookup, create another credential that uses Hashicorp Vault lookup.

Additional resources

For more detail about the LDAP authentication method and its fields, see the [Vault documentation for LDAP auth method](#).

For more information about AppRole authentication method and its fields, see the [Vault documentation for AppRole auth method](#).

For more information about the userpass authentication method and its fields, see the [Vault documentation for userpass auth method](#).

For more information about the Kubernetes auth method and its fields, see the [Vault documentation for Kubernetes auth method](#).

For more information about the TLS certificate auth method and its fields, see the [Vault documentation for TLS certificates auth method](#).

13.2.7. HashiCorp Vault Signed SSH

When you select **HashiCorp Vault Signed SSH** for **Credential Type**, give the following metadata to configure your lookup:

- **Server URL** (required): give the URL used for communicating with HashiCorp Signed SSH's secret management system.
- **Token:** specify the access token used to authenticate HashiCorp's server.
- **CA Certificate:** specify the CA certificate used to verify HashiCorp's server.
- **AppRole role_id:** specify the ID for AppRole authentication.
- **AppRole secret_id:** specify the corresponding secret ID for AppRole authentication.
- **Client Certificate:** specify a PEM-encoded client certificate when using the TLS authentication method, including any required intermediate certificates expected by Hashicorp Vault.
- **Client Certificate Key:** specify a PEM-encoded certificate private key when using the TLS authentication method.
- **TLS Authentication Role:** specify the role or certificate name in Hashicorp Vault that

corresponds to your client certificate when using the TLS authentication method. If it is not provided, Hashicorp Vault attempts to match the certificate automatically.

- **Namespace name:** specify the Namespace name (Hashicorp Vault enterprise only).
- **Kubernetes role:** specify the role name when using Kubernetes authentication.
- **Username:** enter the username of the user to be used to authenticate this service.
- **Password:** enter the password associated with the user to be used to authenticate this service.
- **Path to Auth:** specify a path if other than the default path of `/approle`.

Additional resources

For more information about AppRole authentication method and its fields, see the [Vault documentation for AppRole Auth Method](#).

For more information about the Kubernetes authentication method and its fields, see the [Vault documentation for Kubernetes auth method](#).

For more information about the TLS certificate auth method and its fields, see the [Vault documentation for TLS certificates auth method](#).

13.2.8. Microsoft Azure Key Vault

When you select **Microsoft Azure Key Vault** for **Credential Type**, give the following metadata to configure your lookup:

- **Vault URL (DNS Name)** (required): give the URL used for communicating with Microsoft Azure's key management system
- **Client ID** (required): give the identifier as obtained by Microsoft Entra ID
- **Client Secret** (required): give the secret as obtained by Microsoft Entra ID
- **Tenant ID** (required): give the unique identifier that is associated with an Microsoft Entra ID instance within an Azure subscription
- **Cloud Environment:** select the applicable cloud environment to apply

13.2.9. Thycotic DevOps Secrets Vault

When you select **Thycotic DevOps Secrets Vault** for **Credential Type**, give the following metadata to configure your lookup:

- **Tenant** (required): give the URL used for communicating with Thycotic's secret management system
- **Top-level Domain (TLD):** give the top-level domain designation, for example .com, .edu, or .org, associated with the secret vault you want to integrate
- **Client ID** (required): give the identifier as obtained by the Thycotic secret management system
- **Client Secret** (required): give the secret as obtained by the Thycotic secret management system

13.2.10. Thycotic Secret Server

When you select **Thycotic Secrets Server** for **Credential Type**, give the following metadata to configure your lookup:

- **Secret Server URL** (required): give the URL used for communicating with the Thycotic Secrets Server management system
- **Username** (required): specify the authenticated user for this service
- **Domain**: give the (application) user domain
- **Password** (required): give the password associated with the user

13.2.11. Configuring a GitHub App Installation Access Token Lookup

With this plugin you can use a private GitHub App RSA key as a credential input source to pull access tokens from GitHub App installations. Platform gateway uses existing GitHub authorization from organizations' GitHub repositories.

For more information, see [Generating an installation access token for a GitHub App](#).

Procedure

1. Create a lookup credential that stores your secrets. For more information, see [Creating new credentials](#).
2. Select **GitHub App Installation Access Token Lookup** for **Credential type**, and enter the following attributes to properly configure your lookup:
 - **GitHub App ID**: Enter the App ID provided by your instance of GitHub, this is what is used to authenticate.
 - **GitHub App Installation ID**: Enter the ID of the application into your target organization where the access token is scoped. You must set it up to have access to your target repository.
 - **RSA Private Key**: Enter the generated private key that your GitHub instance generated. You can get it from the GitHub App maintainer within GitHub. For more information, see [Managing private keys for GitHub Apps](#).
3. Click [**Create credential**] to confirm and save the credential.

The following is an example of a configured **GitHub App Installation Access Token Lookup** credential:

Create credential

Name *

github-app-token-lookup

Description

private repo token

Organization

Default

Credential type *

GitHub App Installation Access Token Lookup

Type Details

GitHub API endpoint URL ?

https://api.github.com

GitHub App ID * ?

146130

GitHub App Installation ID * ?

6704280

RSA Private Key * ?

512b-rsa-example-keypair.pem

Browse...

Clear

-----BEGIN RSA PRIVATE KEY-----

MIIBOwIBAAJBAJv8ZpB5hEK7qxP9K3v43hUS5fGT4waKe7ix4

[46mGULBw-cs7W0F

hC7hsugGuaHdPzJ2H4wzjgPMBvPBBwJwCAsEANGJAGGz3e


hC7HMFQNGGuaJ0z

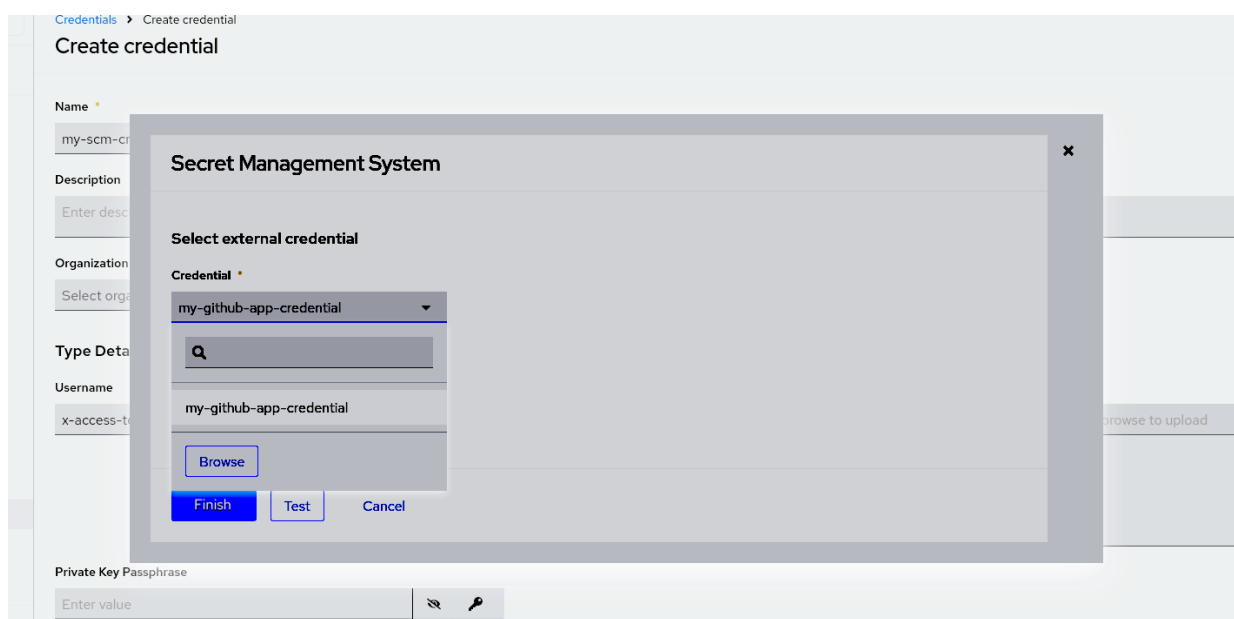
Create credential

Test

Cancel

4. Create a target credential that searches for the lookup credential. To use your lookup in a private repository, select **Source Control** as your **Credential type**. Enter the following attributes to properly configure your target credential:

- **Username:** Enter the username **x-access-token**.
- **Password:** Click the  icon for managing external credentials in the input field. You are prompted to set the input source to use to retrieve your secret information. This is the lookup credential that you have already created.



The screenshot shows the 'Create credential' form with a modal window titled 'Secret Management System'. The modal has a close button (X) in the top right corner. Inside the modal, there is a section 'Select external credential' with a dropdown menu showing 'my-github-app-credential'. Below the dropdown is a search bar with a magnifying glass icon. At the bottom of the modal are three buttons: 'Finish', 'Test', and 'Cancel'. The background form is partially visible, showing fields for 'Name', 'Description', 'Organization', 'Type Details', 'Username', and 'Private Key Passphrase'.

5. Enter an optional description for the metadata requested and click **[Finish]**.

6. Click **[Create credential]** to confirm and save the credential.
7. Verify both your lookup credential and your target credential are now available on the **Credentials** list view. To use the target credential in a project, create a project and enter the following information:
 - **Name:** Enter the name for your project.
 - **Organization:** Select the name of the organization from the drop-down menu..
 - **Execution environment:** Optionally select an execution environment, if applicable.
 - **Source control type:** If you are syncing with a private repository, select **Git** for your source control.

The **Type Details** view opens for additional input. Enter the following information:

- **Source control URL:** Enter the URL of the private repository you want to access. The other related fields pertaining to **branch/tag/commit** and **refspec** are not relevant for use with a lookup credential.
- **Source control credential:** Select the target credential that you have already created.

The following is an example of a configured target credential in a project:

Projects > Create project

Create project

Name *

private repo sync with org credential

Description

Enter description

Organization * **Execution environment** **Source control type ***

Default Select execution environment Git

Content signature validation credential ⓘ

Select content signature validation credential...

Type Details

Source control URL * ⓘ **Source control branch/tag/commit** ⓘ **Source control refspec** ⓘ

https://github.com/arestlelansibletest/test-cred... Enter source control branch/tag/commit Enter source control refspec

Source control credential

scm-github-app-token X

Options

☐ Clean ⓘ ☐ Delete ⓘ ☐ Track submodules ⓘ

☐ Update revision on launch ⓘ ☐ Allow branch override ⓘ

Save project Cancel

8. Click **[Create project]** and the project sync automatically starts. The project **Details** tab displays the progress of the job:

Name

private repo sync with org credential

Organization

Default

Last job status

✓ Success

Source control type

Git

Source control URL ⓘ

https://github.com/arestlelansibletest/test-credentials

Source control credential

Scm: scm-github-app-token

Cache timeout ⓘ

0 seconds

Project base path ⓘ

/home/ansible/aap/controller/data/projects

Playbook directory ⓘ

_173__private_repo_sync_with_org_credential

Created

2/27/2025, 1:19:32 PM by admin

Last modified

2/27/2025, 1:19:32 PM by admin

Troubleshooting

If your project sync fails, you might have to manually re-enter <https://api.github.com> in the **GitHub API endpoint URL** field from Step 2 and re-run your project sync.

14. Secret handling and connection security

Automation controller handles secrets and connections securely.

14.1. Secret handling

Automation controller manages three sets of secrets:

- User passwords for local automation controller users.
- Secrets for automation controller operational use, such as database password or message bus password.
- Secrets for automation use, such as SSH keys, cloud credentials, or external password vault credentials.

NOTE

You must have 'local' user access for the following users:

- postgres
- awx
- redis
- receptor
- nginx

14.1.1. User passwords for local users

Automation controller hashes local automation controller user passwords with the PBKDF2 algorithm using a SHA256 hash. Users who authenticate by external account mechanisms, such as LDAP, SAML, and OAuth, do not have any password or secret stored.

14.1.2. Secret handling for operational use

The operational secrets found in automation controller are as follows:

- `/etc/tower/SECRET_KEY`: A secret key used for encrypting automation secrets in the database. If the `SECRET_KEY` changes or is unknown, you cannot access encrypted fields in the database.
- `/etc/tower/tower.{cert,key}`: An SSL certificate and key for the automation controller web service. A self-signed certificate or key is installed by default; you can provide a locally appropriate certificate and key.
- A database password in `/etc/tower/conf.d/postgres.py` and a message bus password in `/etc/tower/conf.d/channels.py`.

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

If you need to hide these secrets, the files that these secrets are read from are interpreted by Python. You can adjust these files to retrieve these secrets by some other mechanism anytime a service restarts. This is a customer provided modification that might need to be reapplied after every upgrade. Red Hat Support and Red Hat Consulting have examples of such modifications.

NOTE

If the secrets system is down, automation controller cannot get the information and can fail in a way that is recoverable once the service is restored. Using some redundancy on that system is highly recommended.

If you believe the `SECRET_KEY` that automation controller generated for you has been compromised and needs to be regenerated, you can run a tool from the installer that behaves much like the automation controller backup and restore tool.

IMPORTANT

Ensure that you backup your automation controller database before you generate a new secret key.

To generate a new secret key:

1. Follow the procedure described in the [Backing up and Restoring](#) section.
2. Use the inventory from your install (the same inventory with which you run backups and restores), and run the following command:

```
setup.sh -k.
```

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

14.1.3. Secret handling for automation use

Automation controller stores a variety of secrets in the database that are either used for automation or are a result of automation.

These secrets include the following:

- All secret fields of all credential types, including passwords, secret keys, authentication tokens, and secret cloud credentials.
- Secret tokens and passwords for external services defined automation controller settings.
- "password" type survey field entries.

To encrypt secret fields, automation controller uses AES in CBC mode with a 256-bit key for encryption, PKCS7 padding, and HMAC using SHA256 for authentication.

The encryption or decryption process derives the AES-256 bit encryption key from the `SECRET_KEY`, the field name of the model field and the database assigned auto-incremented record ID. Therefore, if any attribute used in the key generation process changes, the automation controller fails to correctly decrypt the secret.

Automation controller is designed so that:

- The `SECRET_KEY` is never readable in playbooks that automation controller launches.
- These secrets are never readable by automation controller users.
- No secret field values are ever made available by the automation controller REST API.

If a secret value is used in a playbook, it is recommended that you use `no_log` on the task so that it is not accidentally logged.

14.1.4. Connection security

Automation controller allows for connections to internal services, external access, and managed nodes.

NOTE

You must have 'local' user access for the following users:

- postgres
- awx
- redis
- receptor
- nginx

14.1.5. Internal services

Automation controller connects to the following services as part of internal operation:

PostgreSQL database

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

A Redis key or value store

The connection to Redis is over a local UNIX socket, restricted to the awx service user.

14.1.6. External access

Automation controller is accessed using standard HTTP/HTTPS on standard ports, provided by Nginx. A self-signed certificate or key is installed by default; you can provide a locally appropriate certificate and key. SSL/TLS algorithm support is configured in the `/etc/nginx/nginx.conf` configuration file. An "intermediate" profile is used by default, that you can configure. You must reapply changes after each update.

14.1.7. Managed nodes

Automation controller connects to managed machines and services as part of automation. All connections to managed machines are done by standard secure mechanisms, such as SSH, WinRM, or SSL/TLS.

Each of these inherits configuration from the system configuration for the feature in question, such as the system OpenSSL configuration.

15. Security best practices

You can deploy automation controller to automate typical environments securely. However, managing certain operating system environments, automation, and automation platforms, can require additional best practices to ensure security.

To secure Red Hat Enterprise Linux start with the following release-appropriate security guide:

- For Red Hat Enterprise Linux 8, see [Security hardening](#).
- For Red Hat Enterprise Linux 9, see [Security hardening](#).

15.1. Understand the architecture of Ansible Automation Platform and automation controller

Ansible Automation Platform and automation controller comprise a general-purpose, declarative automation platform. That means that when an Ansible Playbook is launched (by automation controller, or directly on the command line), the playbook, inventory, and credentials provided to Ansible are considered to be the source of truth. If you want policies around external verification of specific playbook content, job definition, or inventory contents, you must complete these processes before the automation is launched, either by the automation controller web UI, or the automation controller API.

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

At a higher level, tools exist that enable creation of approvals and policy-based actions around

arbitrary workflows, including automation. These tools can then use Ansible through the automation controller's API to perform automation.

You must use a secure default administrator password at the time of automation controller installation. For more information, see [Change the automation controller Administrator Password](#).

Automation controller exposes services on certain well-known ports, such as port 80 for HTTP traffic and port 443 for HTTPS traffic. Do not expose automation controller on the open internet, which reduces the threat surface of your installation.

15.1.1. Granting access

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

- [Minimize administrative accounts](#)
- [Minimize local system access](#)
- [Remove access to credentials from users](#)
- [Enforce separation of duties](#)

15.1.2. Minimize administrative accounts

Minimizing the access to system administrative accounts is crucial for maintaining a secure system. A system administrator or root user can access, edit, and disrupt any system application. Limit the number of people or accounts with root access, where possible. Do not give out *sudo* to *root* or *awx* (the automation controller user) to untrusted users. Note that when restricting administrative access through mechanisms like *sudo*, restricting to a certain set of commands can still give a wide range of access. Any command that enables execution of a shell or arbitrary shell commands, or any command that can change files on the system, is equal to full root access.

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

15.1.3. Minimize local system access

When you use automation controller with best practices, it does not require local user access except for administrative purposes. Non-administrator users do not have access to the automation controller system.

15.1.4. Remove user access to credentials

If an automation controller credential is only stored in the controller, you can further secure it. You can configure services such as OpenSSH to only permit credentials on connections from specific addresses. Credentials used by automation can be different from credentials used by system administrators for disaster-recovery or other ad hoc management, allowing for easier auditing.

15.1.5. Enforce separation of duties

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

15.2. Available resources

Several resources exist in automation controller and elsewhere to ensure a secure platform. Consider using the following functionalities:

- [Existing security functionality](#)
- [External account stores](#)
- [Django password policies](#)

15.2.1. Existing security functionality

Note the following:

- Do not disable SELinux or automation controller's existing multi-tenant containment.
- Use automation controller's role-based access control (RBAC) to delegate the minimum level of privileges required to run automation. For more information, see [Managing access with role based access control](#).
- Use teams in automation controller to assign permissions to groups of users rather than to users individually.

15.2.2. External account stores

Maintaining a full set of users in automation controller can be a time-consuming task in a large organization. Automation controller supports connecting to external account sources by LDAP, SAML 2.0, and certain OAuth providers. Using this eliminates a source of error when working with permissions.

15.2.3. Django password policies

Automation controller administrators can use Django to set password policies at creation time through `AUTH_PASSWORD_VALIDATORS` to validate automation controller user passwords. In the `custom.py` file located at `/etc/tower/conf.d` of your automation controller instance, add the following code block example:

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

```

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

```

Ensure that you restart your automation controller instance for the change to take effect. For more information, see [Start, stop, and restart automation controller](#).

Additional resources

[Password validation](#)

16. The *awx-manage* Utility

Use the **awx-manage** utility to access detailed internal information of automation controller. Commands for **awx-manage** must run as the **awx** user only.

16.1. Inventory Import

awx-manage is a mechanism by which an automation controller administrator can import inventory directly into automation controller.

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

For help with **awx-manage**, run the following command:

```
awx-manage inventory_import [--help]
```

The **inventory_import** command synchronizes an automation controller inventory object with a text-based inventory file, dynamic inventory script, or a directory of one or more, as supported by core Ansible.

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

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

By default, inventory data already stored in automation controller blends with data from the external source.

To use only the external data, specify `--overwrite`.

To specify that any existing hosts get variable data exclusively from the `--source`, specify `--overwrite_vars`.

The default behavior adds any new variables from the external source, overwriting keys that already exist, but preserving any variables that were not sourced from the external data source.

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

NOTE

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

16.2. Cleanup of old data

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

- `awx-manage cleanup_jobs [--help]`

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

- `awx-manage cleanup_activitystream [--help]`

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

16.3. Cluster management

For more information about the `awx-manage provision_instance` and `awx-manage deprovision_instance` commands, see [Clustering](#).

NOTE

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

16.4. Analytics gathering

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

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

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

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

The `--since` parameter is optional.

The `--json` flag specifies the output format and is optional.

17. Backup and restore

You can backup and restore your system by using the Ansible Automation Platform setup playbook.

For more information, see the [Backup and restore clustered environments](#) section.

NOTE

When backing up Ansible Automation Platform, use the installation program that matches your currently installed version of Ansible Automation Platform.

When restoring Ansible Automation Platform, use the latest installation program available at the time of the restore. For example, if you are restoring a backup taken from version 2.6-1, use the latest 2.6-x installation program available at the time of the restore.

Backup and restore functionality only works with the PostgreSQL versions supported by your current Ansible Automation Platform version. For more information, see [System requirements](#) in *Planning your installation*.

The Ansible Automation Platform setup playbook is invoked as `setup.sh` from the path where you unpacked the platform installer tar file. It uses the same inventory file used by the install playbook. The setup script takes the following arguments for backing up and restoring:

- `-b`: Perform a database backup rather than an installation.
- `-r`: Perform a database restore rather than an installation.

As the root user, call `setup.sh` with the appropriate parameters and the Ansible Automation Platform backup or restored as configured:

```
root@localhost:~# ./setup.sh -b
root@localhost:~# ./setup.sh -r
```

Backup files are created on the same path that `setup.sh` script exists. You can change it by specifying the following `EXTRA_VARS`:

```
root@localhost:~# ./setup.sh -e 'backup_dest=/path/to/backup_dir/' -b
```

A default restore path is used unless you provide `EXTRA_VARS` with a non-default path, as shown in the following example:

```
root@localhost:~# ./setup.sh -e
'restore_backup_file=/path/to/nondefault/backup.tar.gz' -r
```

Optionally, you can override the inventory file used by passing it as an argument to the setup script:

```
setup.sh -i <inventory file>
```

17.1. Backup and restore playbooks

In addition to the `install.yml` file included with your `setup.sh` setup playbook, there are also `backup.yml` and `restore.yml` files.

These playbooks serve to backup and restore.

- The overall backup, backs up:
 - The database
 - The `SECRET_KEY` file
- The per-system backups include:
 - Custom configuration files
 - Manual projects
- The restore backup restores the backed up files and data to a freshly installed and working second instance of automation controller.

When restoring your system, installation program checks to see that the backup file exists before beginning the restoration. If the backup file is not available, your restoration fails.

NOTE

Make sure that your automation controller hosts are properly set up with SSH keys, user or pass variables in the hosts file, and that the user has `sudo` access.

17.2. Backup and restoration considerations

Consider the following points when you backup and restore your system:

Disk space

Review your disk space requirements to ensure you have enough room to backup configuration files, keys, other relevant files, and the database of the Ansible Automation Platform installation.

System credentials

Confirm you have the required system credentials when working with a local database or a remote database. On local systems, you might need `root` or `sudo` access, depending on how credentials are set up. On remote systems, you might need different credentials to grant you access to the remote system you are trying to backup or restore.

NOTE

The Ansible Automation Platform database backups are staged on each node at `/var/backups/automation-platform` through the variable `backup_dir`. You might need to mount a new volume to `/var/backups` or change the staging location with the variable `backup_dir` to prevent issues with disk space before running the `./setup.sh -b` script.

Version

You must always use the most recent minor version of a release to backup or restore your Ansible Automation Platform installation version. For example, if the current platform version you are on is 2.0.x, only use the latest 2.0 installer.

File path

When using `setup.sh` to do a restore from the default restore file path, `/var/lib/awx`, `-r` is still required to do the restore, but it no longer accepts an argument. If a non-default restore file path is needed, you must provide this as an extra_var (`root@localhost:~# ./setup.sh -e 'restore_backup_file=/path/to/nondefault/backup.tar.gz' -r`).

Directory

If the backup file is placed in the same directory as the `setup.sh` installer, the restore playbook automatically locates the restore files. In this case, you do not need to use the `restore_backup_file` extra var to specify the location of the backup file.

17.3. Backup and restore clustered environments

The procedure for backup and restore for a clustered environment is similar to a single install, except for some of the following considerations:

NOTE

For more information about installing clustered environments, see the [Install and configure](#) section.

- If restoring to a new cluster, ensure that the old cluster is shut down before proceeding because they can conflict with each other when accessing the database.
- Per-node backups are only restored to nodes bearing the same hostname as the backup.
- When restoring to an existing cluster, the restore has the following:
 - A dump of the PostgreSQL database
 - UI artifacts, included in the database dump
 - An automation controller configuration (retrieved from `/etc/tower`)
 - An automation controller secret key
 - Manual projects

17.3.1. Restore to a different cluster

When restoring a backup to a separate instance or cluster, manual projects and custom settings under `/etc/tower` are retained. Job output and job events are stored in the database, and therefore, not affected.

The restore process does not alter instance groups present before the restore. It does not introduce any new instance groups either. Restored automation controller resources that were associated to instance groups likely need to be reassigned to instance groups present on the new automation controller cluster.

18. Usability Analytics and Data Collection

Usability data collection is included with automation controller to collect data to better understand how automation controller users interact with it.

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

Automation controller collects user data automatically to help improve the product.

For information about setting up Automation Analytics, see [Configuring Automation Analytics](#).

18.1. Automation Analytics

When you imported your license for the first time, you were automatically opted in for the collection of data that powers Automation Analytics, a cloud service that is part of the Ansible Automation Platform subscription.

IMPORTANT

For opt-in of Automation Analytics to have any effect, your instance of automation controller must be running on Red Hat Enterprise Linux.

As with Red Hat Insights, Automation Analytics is built to collect the minimum amount of data needed. No credential secrets, personal data, automation variables, or task output is gathered.

When you imported your license for the first time, you were automatically opted in to Automation Analytics. To configure or disable this feature, see [Configuring Automation Analytics](#).

By default, the data is collected every four hours. When you enable this feature, data is collected up to a month in arrears (or until the previous collection). You can turn off this data collection at any time in the **Miscellaneous System settings** of the System configuration window.

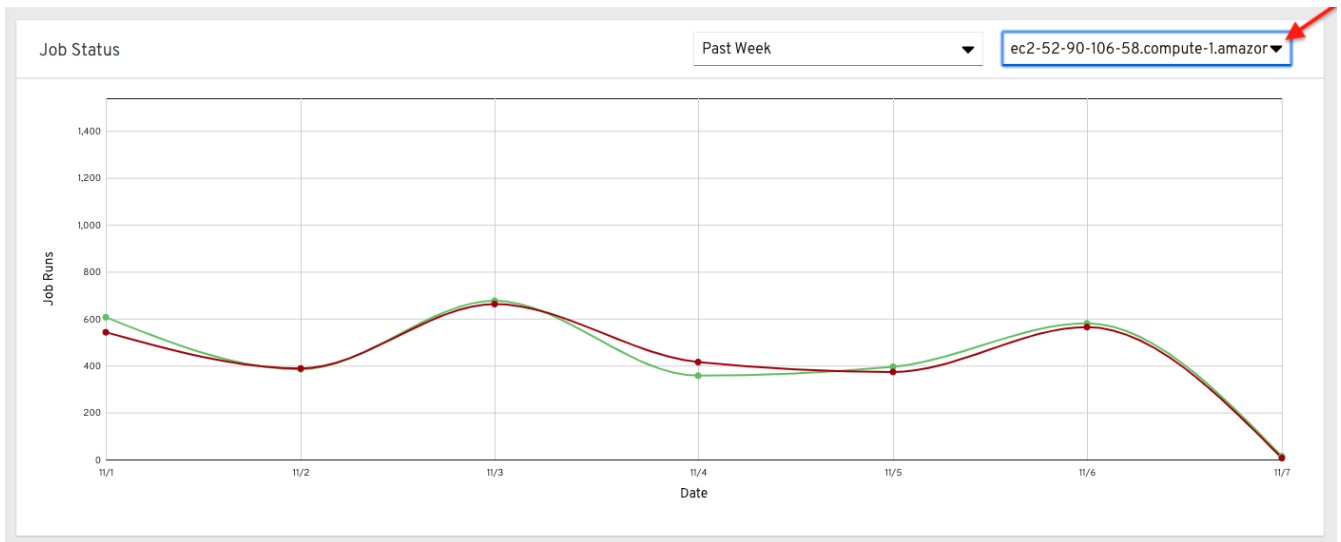
This setting can also be enabled through the API by specifying `INSIGHTS_TRACKING_STATE = true` in either of these endpoints:

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

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

Clusters data is the default view. This graph represents the number of job runs across all automation controller clusters over a period of time. The previous example shows a span of a week in a stacked bar-style chart that is organized by the number of jobs that ran successfully (in green) and jobs that failed (in red).

Alternatively, you can select a single cluster to view its job status information.



This multi-line chart represents the number of job runs for a single automation controller cluster for a specified period of time. The preceding example shows a span of a week, organized by the number of successfully running jobs (in green) and jobs that failed (in red). You can specify the number of successful and failed job runs for a selected cluster over a span of one week, two weeks, and monthly increments.

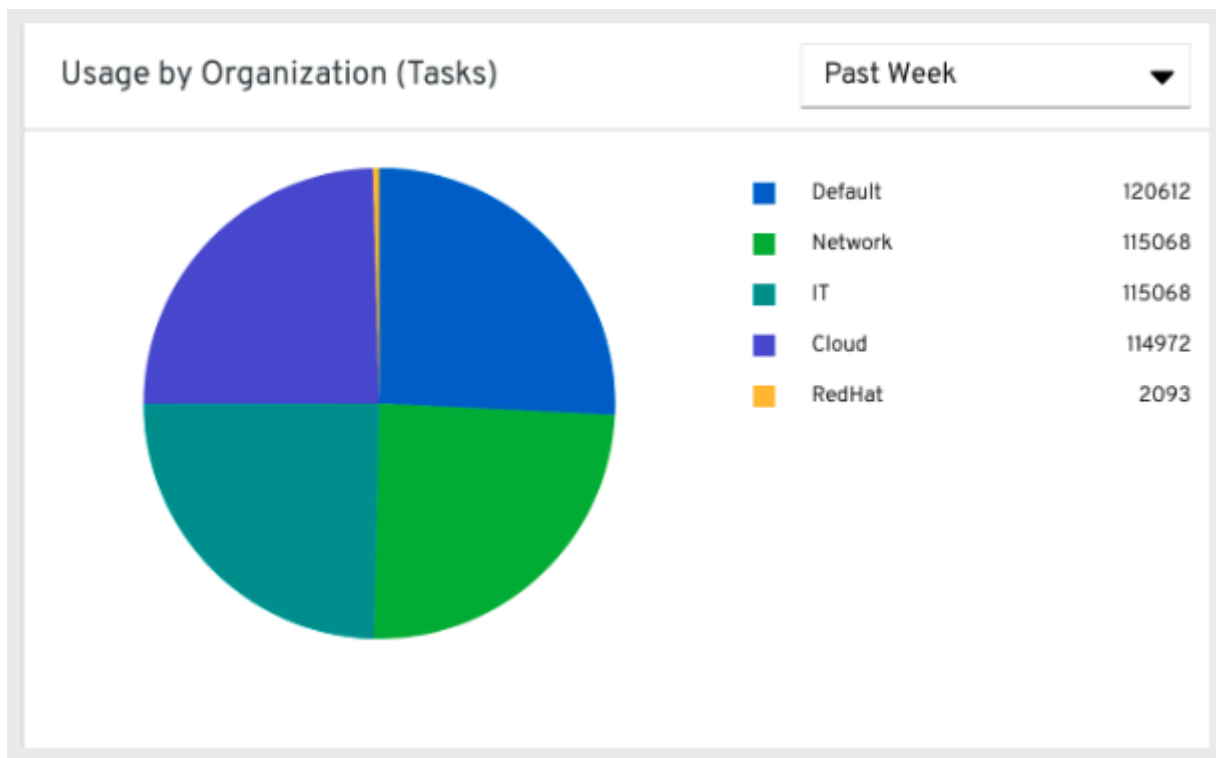
On the clouds navigation panel, select **Organization Statistics** to view information for the following:

- [Use by organization](#)
- [Job runs by organization](#)
- [Organization status](#)

NOTE The organization statistics page will be deprecated in a future release.

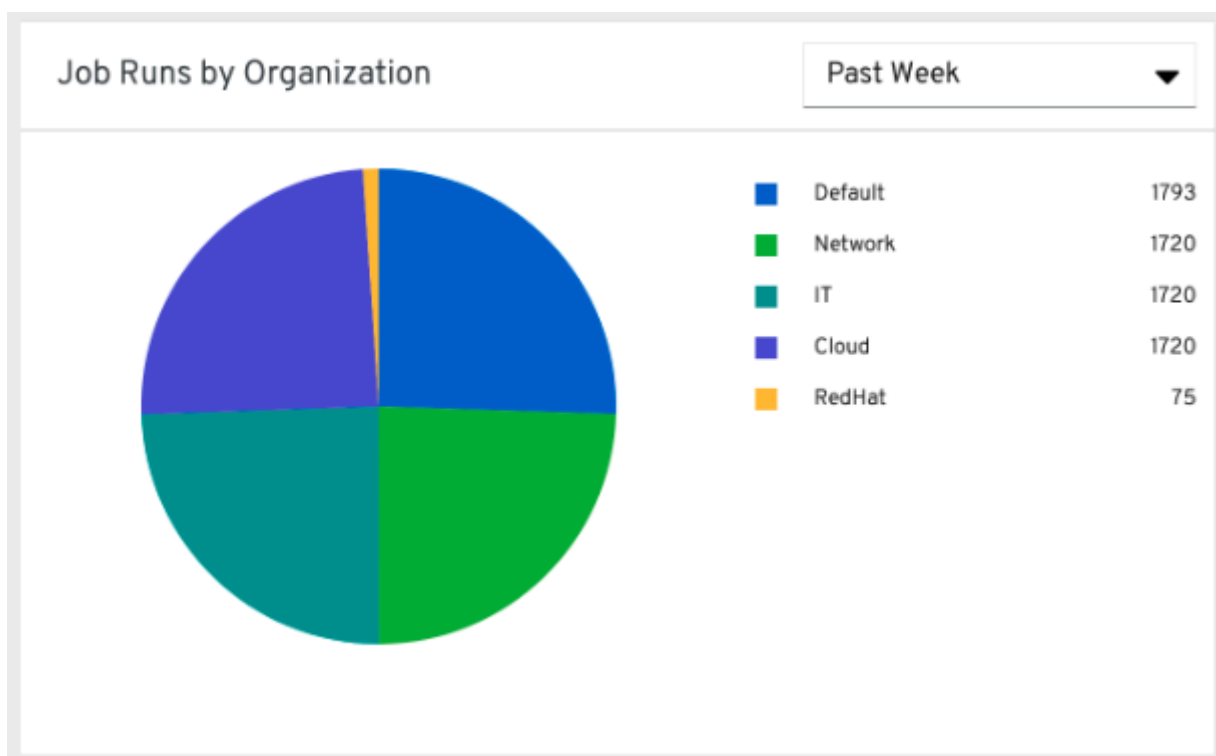
18.1.1. Use by organization

The following chart represents the number of tasks run inside all jobs by a particular organization.



18.1.2. Job runs by organization

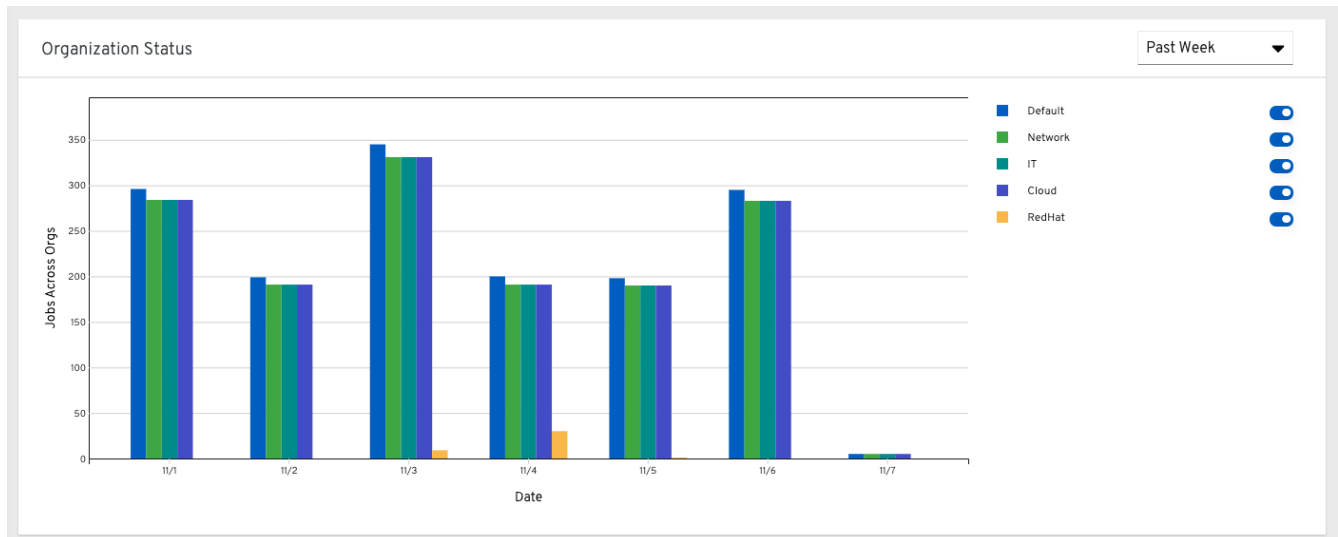
This chart represents automation controller use across all automation controller clusters by organization, calculated by the number of jobs run by that organization.



18.1.3. Organization status

This bar chart represents automation controller use by organization and date, which is calculated by the number of jobs run by that organization on a particular date.

Alternatively, you can specify to show the number of job runs per organization in one week, two weeks, and monthly increments.



18.2. Details of data collection

Automation Analytics collects the following classes of data from automation controller:

- Basic configuration, such as which features are enabled, and what operating system is being used
- Topology and status of the automation controller environment and hosts, including capacity and health
- Counts of automation resources:
 - organizations, teams, and users
 - inventories and hosts
 - credentials (indexed by type)
 - projects (indexed by type)
 - templates
 - schedules
 - active sessions
 - running and pending jobs
- Job execution details (start time, finish time, launch type, and success)
- Automation task details (success, host id, playbook/role, task name, and module used)

You can use `awx-manage gather_analytics` (without `--ship`) to inspect the data that automation controller sends, so that you can satisfy your data collection concerns. This creates a .tar file that contains the analytics data that is sent to Red Hat.

This file contains several JSON and CSV files. Each file contains a different set of analytics data.

- [manifest.json](#)

- [config.json](#)
- [instance_info.json](#)
- [counts.json](#)
- [org_counts.json](#)
- [cred_type_counts.json](#)
- [inventory_counts.json](#)
- [projects_by_scm_type.json](#)
- [query_info.json](#)
- [job_counts.json](#)
- [job_instance_counts.json](#)
- [unified_job_template_table.csv](#)
- [unified_jobs_table.csv](#)
- [workflow_job_template_node_table.csv](#)
- [workflow_job_node_table.csv](#)
- [events_table.csv](#)

18.2.1. manifest.json

manifest.json is the manifest of the analytics data. It describes each file included in the collection, and what version of the schema for that file is included.

The following is an example **manifest.json** file:

```
"config.json": "1.1",
"counts.json": "1.0",
"cred_type_counts.json": "1.0",
"events_table.csv": "1.1",
"instance_info.json": "1.0",
"inventory_counts.json": "1.2",
"job_counts.json": "1.0",
"job_instance_counts.json": "1.0",
"org_counts.json": "1.0",
"projects_by_scm_type.json": "1.0",
"query_info.json": "1.0",
"unified_job_template_table.csv": "1.0",
"unified_jobs_table.csv": "1.0",
"workflow_job_node_table.csv": "1.0",
"workflow_job_template_node_table.csv": "1.0"
}
```

18.2.2. config.json

The config.json file contains a subset of the configuration endpoint `/api/v2/config` from the cluster. An example config.json is:

```
{
  "ansible_version": "2.9.1",
  "authentication_backends": [
    "social_core.backends.azuread.AzureADOAuth2",
    "django.contrib.auth.backends.ModelBackend"
  ],
  "external_logger_enabled": true,
  "external_logger_type": "splunk",
  "free_instances": 1234,
  "install_uuid": "d3d497f7-9d07-43ab-b8de-9d5cc9752b7c",
  "instance_uuid": "bed08c6b-19cc-4a49-bc9e-82c33936e91b",
  "license_expiry": 34937373,
  "license_type": "enterprise",
  "logging_aggregators": [
    "awx",
    "activity_stream",
    "job_events",
    "system_tracking"
  ],
  "pendo_tracking": "detailed",
  "platform": {
    "dist": [
      "redhat",
      "7.4",
      "Maipo"
    ],
    "release": "3.10.0-693.el7.x86_64",
    "system": "Linux",
    "type": "traditional"
  },
  "total_licensed_instances": 2500,
  "controller_url_base": "https://ansible.rhdemo.io",
  "controller_version": "3.6.3"
}
```

Which includes the following fields:

- **ansible_version:** The system Ansible version on the host
- **authentication_backends:** The user authentication backends that are available. For more information, see [Configuring an authentication type](#).
- **external_logger_enabled:** Whether external logging is enabled
- **external_logger_type:** What logging backend is in use if enabled. For more information, see [Logging and aggregation](#).

- **logging_aggregators:** What logging categories are sent to external logging. For more information, see [Logging and aggregation](#).
- **free_instances:** How many hosts are available in the license. A value of zero means the cluster is fully consuming its license.
- **install_uuid:** A UUID for the installation (identical for all cluster nodes)
- **instance_uuid:** A UUID for the instance (different for each cluster node)
- **license_expiry:** Time to expiry of the license, in seconds
- **license_type:** The type of the license (should be 'enterprise' for most cases)
- **pendo_tracking:** State of [usability_data_collection](#)
- **platform:** The operating system the cluster is running on
- **total_licensed_instances:** The total number of hosts in the license
- **controller_url_base:** The base URL for the cluster used by clients (shown in Automation Analytics)
- **controller_version:** Version of the software on the cluster

18.2.3. instance_info.json

The [instance_info.json](#) file contains detailed information on the instances that make up the cluster, organized by instance UUID.

The following is an example [instance_info.json](#) file:

```
{
  "bed08c6b-19cc-4a49-bc9e-82c33936e91b": {
    "capacity": 57,
    "cpu": 2,
    "enabled": true,
    "last_isolated_check": "2019-08-15T14:48:58.553005+00:00",
    "managed_by_policy": true,
    "memory": 8201400320,
    "uuid": "bed08c6b-19cc-4a49-bc9e-82c33936e91b",
    "version": "3.6.3"
  }
  "c0a2a215-0e33-419a-92f5-e3a0f59bfaee": {
    "capacity": 57,
    "cpu": 2,
    "enabled": true,
    "last_isolated_check": "2019-08-15T14:48:58.553005+00:00",
    "managed_by_policy": true,
    "memory": 8201400320,
    "uuid": "c0a2a215-0e33-419a-92f5-e3a0f59bfaee",
    "version": "3.6.3"
  }
}
```

Which includes the following fields:

- **capacity**: The capacity of the instance for executing tasks.
- **cpu**: Processor cores for the instance
- **memory**: Memory for the instance
- **enabled**: Whether the instance is enabled and accepting tasks
- **managed_by_policy**: Whether the instance's membership in instance groups is managed by policy, or manually managed
- **version**: Version of the software on the instance

18.2.4. counts.json

The `counts.json` file contains the total number of objects for each relevant category in a cluster.

The following is an example `counts.json` file:

```
{
  "active_anonymous_sessions": 1,
  "active_host_count": 682,
  "active_sessions": 2,
  "active_user_sessions": 1,
  "credential": 38,
  "custom_inventory_script": 2,
  "custom_virtualenvs": 4,
  "host": 697,
  "inventories": {
    "normal": 20,
    "smart": 1
  },
  "inventory": 21,
  "job_template": 78,
  "notification_template": 5,
  "organization": 10,
  "pending_jobs": 0,
  "project": 20,
  "running_jobs": 0,
  "schedule": 16,
  "team": 5,
  "unified_job": 7073,
  "user": 28,
  "workflow_job_template": 15
}
```

Each entry in this file is for the corresponding API objects in `/api/v2`, with the exception of the active session counts.

18.2.5. org_counts.json

The `org_counts.json` file contains information on each organization in the cluster, and the number of users and teams associated with that organization.

The following is an example `org_counts.json` file:

```
{
  "1": {
    "name": "Operations",
    "teams": 5,
    "users": 17
  },
  "2": {
    "name": "Development",
    "teams": 27,
    "users": 154
  },
  "3": {
    "name": "Networking",
    "teams": 3,
    "users": 28
  }
}
```

18.2.6. cred_type_counts.json

The `cred_type_counts.json` file has information about different credential types in the cluster, and how many credentials exist for each type.

The following is an example `cred_type_counts.json` file:

```
{
  "1": {
    "credential_count": 15,
    "managed_by_controller": true,
    "name": "Machine"
  },
  "2": {
    "credential_count": 2,
    "managed_by_controller": true,
    "name": "Source Control"
  },
  "3": {
    "credential_count": 3,
    "managed_by_controller": true,
    "name": "Vault"
  },
  "4": {
```

```

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

```

18.2.7. inventory_counts.json

The `inventory_counts.json` file contains information on the different inventories in the cluster.

The following is an example `inventory_counts.json` file:

```

{
  "1": {
    "hosts": 211,
    "kind": "",
    "name": "AWS Inventory",
    "source_list": [
      {
        "name": "AWS",
        "num_hosts": 211,
        "source": "ec2"
      }
    ],
    "sources": 1
  },
  "2": {
    "hosts": 15,
    "kind": "",
    "name": "Manual inventory",
    "source_list": [],
    "sources": 0
  },
  "3": {
    "hosts": 25,
    "kind": "",
    "name": "SCM inventory - test repo",
    "source_list": [
      {
        "name": "Git source",
        "num_hosts": 25,

```



```

        "source": "scm"
      }
    ],
    "sources": 1
  }
  "4": {
    "num_hosts": 5,
    "kind": "smart",
    "name": "Filtered AWS inventory",
    "source_list": [],
    "sources": 0
  }
}

```

18.2.8. projects_by_scm_type.json

The `projects_by_scm_type.json` file provides a breakdown of all projects in the cluster, by source control type.

The following is an example `projects_by_scm_type.json` file:

```

{
  "git": 27,
  "hg": 0,
  "insights": 1,
  "manual": 0,
  "svn": 0
}

```

18.2.9. query_info.json

The `query_info.json` file provides details on when and how the data collection happened.

The following is an example `query_info.json` file:

```

{
  "collection_type": "manual",
  "current_time": "2019-11-22 20:10:27.751267+00:00",
  "last_run": "2019-11-22 20:03:40.361225+00:00"
}

```

`collection_type` is one of `manual` or `automatic`.

18.2.10. job_counts.json

The `job_counts.json` file provides details on the job history of the cluster, describing both how jobs were launched, and what their finishing status is.

The following is an example `job_counts.json` file:

```
"launch_type": {
  "dependency": 3628,
  "manual": 799,
  "relaunch": 6,
  "scheduled": 1286,
  "scm": 6,
  "workflow": 1348
},
"status": {
  "canceled": 7,
  "failed": 108,
  "successful": 6958
},
"total_jobs": 7073
}
```

18.2.11. job_instance_counts.json

The `job_instance_counts.json` file provides the same detail as `job_counts.json`, broken down by instance.

The following is an example `job_instance_counts.json` file:

```
{
  "localhost": {
    "launch_type": {
      "dependency": 3628,
      "manual": 770,
      "relaunch": 3,
      "scheduled": 1009,
      "scm": 6,
      "workflow": 1336
    },
    "status": {
      "canceled": 2,
      "failed": 60,
      "successful": 6690
    }
  }
}
```

Note that instances in this file are by hostname, not by UUID as they are in `instance_info`.

18.2.12. unified_job_template_table.csv

The `unified_job_template_table.csv` file provides information on job templates in the system. Each

line contains the following fields for the job template:

- **id**: Job template id.
- **name**: Job template name.
- **polymorphic_ctype_id**: The id of the type of template it is.
- **model**: The name of the `polymorphic_ctype_id` for the template. Examples include `project`, `systemjobtemplate`, `jobtemplate`, `inventorysource`, and `workflowjobtemplate`.
- **created**: When the template was created.
- **modified**: When the template was last updated.
- **created_by_id**: The `userid` that created the template. Blank if done by the system.
- **modified_by_id**: The `userid` that last modified the template. Blank if done by the system.
- **current_job_id**: Currently executing job id for the template, if any.
- **last_job_id**: Last execution of the job.
- **last_job_run**: Time of last execution of the job.
- **last_job_failed**: Whether the `last_job_id` failed.
- **status**: Status of `last_job_id`.
- **next_job_run**: Next scheduled execution of the template, if any.
- **next_schedule_id**: Schedule id for `next_job_run`, if any.

18.2.13. unified_jobs_table.csv

The `unified_jobs_table.csv` file provides information on jobs run by the system.

Each line contains the following fields for a job:

- **id**: Job id.
- **name**: Job name (from the template).
- **polymorphic_ctype_id**: The id of the type of job it is.
- **model**: The name of the `polymorphic_ctype_id` for the job. Examples include `job` and `workflow`.
- **organization_id**: The organization ID for the job.
- **organization_name**: Name for the `organization_id`.
- **created**: When the job record was created.
- **started**: When the job started executing.
- **finished**: When the job finished.
- **elapsed**: Elapsed time for the job in seconds.
- **unified_job_template_id**: The template for this job.
- **launch_type**: One of `manual`, `scheduled`, `relaunched`, `scm`, `workflow`, or `dependency`.
- **schedule_id**: The id of the schedule that launched the job, if any,

- **instance_group_id**: The instance group that executed the job.
- **execution_node**: The node that executed the job (hostname, not UUID).
- **controller_node**: The automation controller node for the job, if run as an isolated job, or in a container group.
- **cancel_flag**: Whether the job was canceled.
- **status**: Status of the job.
- **failed**: Whether the job failed.
- **job_explanation**: Any additional detail for jobs that failed to execute properly.
- **forks**: Number of forks executed for this job.

18.2.14. workflow_job_template_node_table.csv

The `workflow_job_template_node_table.csv` file provides information on the nodes defined in workflow job templates on the system.

Each line contains the following fields for a workflow job template node:

- **id**: Node id.
- **created**: When the node was created.
- **modified**: When the node was last updated.
- **unified_job_template_id**: The id of the job template, project, inventory, or other parent resource for this node.
- **workflow_job_template_id**: The workflow job template that contains this node.
- **inventory_id**: The inventory used by this node.
- **success_nodes**: Nodes that are triggered after this node succeeds.
- **failure_nodes**: Nodes that are triggered after this node fails.
- **always_nodes**: Nodes that always are triggered after this node finishes.
- **all_parents_must_converge**: Whether this node requires all its parent conditions satisfied to start.

18.2.15. workflow_job_node_table.csv

The `workflow_job_node_table.csv` provides information on the jobs that have been executed as part of a workflow on the system.

Each line contains the following fields for a job run as part of a workflow:

- **id**: Node id.
- **created**: When the node was created.
- **modified**: When the node was last updated.
- **job_id**: The job id for the job run for this node.

- **unified_job_template_id**: The id of the job template, project, inventory, or other parent resource for this node.
- **workflow_job_template_id**: The workflow job template that contains this node.
- **inventory_id**: The inventory used by this node.
- **success_nodes**: Nodes that are triggered after this node succeeds.
- **failure_nodes**: Nodes that are triggered after this node fails.
- **always_nodes**: Nodes that always are triggered after this node finishes.
- **do_not_run**: Nodes that were not run in the workflow due to their start conditions not being triggered.
- **all_parents_must_converge**: Whether this node requires all its parent conditions satisfied to start.

18.2.16. events_table.csv

The `events_table.csv` file provides information on all job events from all job runs in the system.

Each line contains the following fields for a job event:

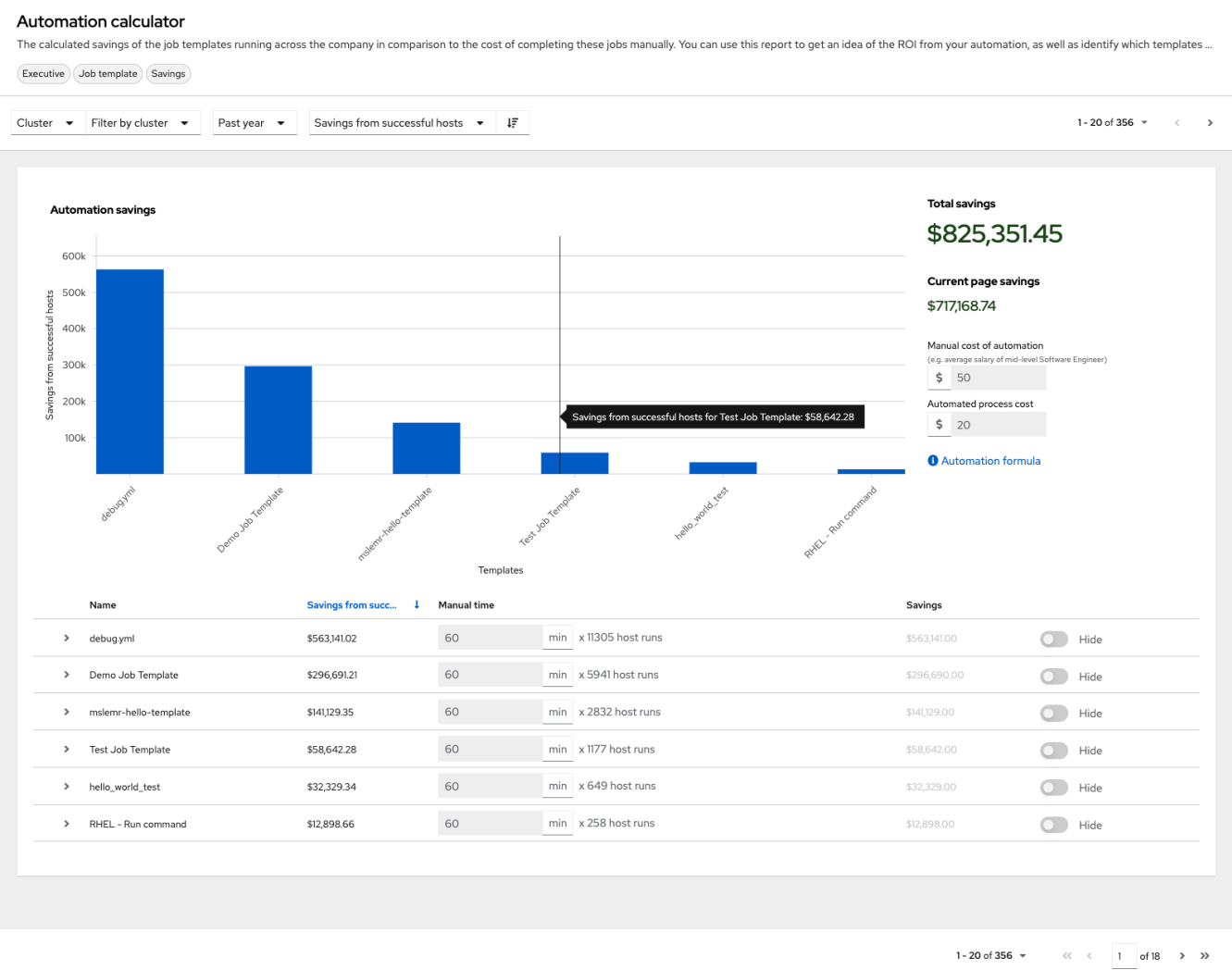
- **id**: Event id.
- **uuid**: Event UUID.
- **created**: When the event was created.
- **parent_uuid**: The parent UUID for this event, if any.
- **event**: The Ansible event type.
- **task_action**: The module associated with this event, if any (such as `command` or `yum`).
- **failed**: Whether the event returned `failed`.
- **changed**: Whether the event returned `changed`.
- **playbook**: Playbook associated with the event.
- **play**: Play name from playbook.
- **task**: Task name from playbook.
- **role**: Role name from playbook.
- **job_id**: Id of the job this event is from.
- **host_id**: Id of the host this event is associated with, if any.
- **host_name**: Name of the host this event is associated with, if any.
- **start**: Start time of the task.
- **end**: End time of the task.
- **duration**: Duration of the task.
- **warnings**: Any warnings from the task or module.
- **deprecations**: Any deprecation warnings from the task or module.

18.3. Analytics Reports

Reports for data collected are available through console.redhat.com.

Other Automation Analytics data currently available and accessible through the platform UI include the following:

Automation Calculator is a view-only version of the Automation Calculator utility that shows a report that represents (possible) savings to the subscriber.



19.1. Unable to login to automation controller through HTTP

Access to automation controller is intentionally restricted through a secure protocol (HTTPS). In cases where your configuration is set up to run an automation controller node behind a load balancer or proxy as "HTTP only", and you only want to access it without SSL (for troubleshooting, for example), you must add the following settings in the `custom.py` file located at `/etc/tower/conf.d` of your automation controller instance:

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

If you change these settings to `false` it enables automation controller to manage cookies and login sessions when using the HTTP protocol. You must do this on every node of a cluster installation.

To apply the changes, run:

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

19.2. Unable to run a job

If you are unable to run a job from a playbook, review the playbook YAML file. When importing a playbook, either manually or by a source control mechanism, keep in mind that the host definition is controlled by automation controller and should be set to `hosts:all`.

19.3. Playbooks do not show up in the Job Template list

If your playbooks are not showing up in the **Job Template** list, check the following:

- Ensure that the playbook is valid YML and can be parsed by Ansible.
- Ensure that the permissions and ownership of the project path (`/var/lib/awx/projects`) is set up so that the "awx" system user can view the files. Run the following command to change the ownership:

```
chown awx -R /var/lib/awx/projects/
```

19.4. Playbook stays in pending

If you are attempting to run a playbook job and it stays in the `Pending` state indefinitely, try the following actions:

- Ensure that all supervisor services are running through `supervisorctl status`.
- Ensure that the `/var/` partition has more than 1 GB of space available. Jobs do not complete

with insufficient space on the `/var/` partition.

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

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

19.5. Reusing an external database causes installations to fail

Instances have been reported where reusing the external database during subsequent installation of nodes causes installation failures.

Example

You perform a clustered installation. Then, you need to do this again and perform a second clustered installation reusing the same external database, only this subsequent installation failed.

When setting up an external database that has been used in a prior installation, you must manually clear the database used for the clustered node before any additional installations can succeed.

19.6. Viewing private EC2 VPC instances in the automation controller inventory

By default, automation controller only shows instances in a VPC that have an Elastic IP (EIP) associated with them.

Procedure

1. From the navigation panel, select **Automation Execution** › **Infrastructure** › **Inventories**.
2. Select the inventory that has the **Source** set to **Amazon EC2**, and click the **Source** tab. In the **Source Variables** field, enter:

```
vpc_destination_variable: private_ip_address
```

3. Click **[Save]** and trigger an update of the group.

Verification

Once you complete these steps, you can see your VPC instances.

NOTE

Automation controller must be running inside the VPC with access to those instances if you want to configure them.

20. Automation controller tips and tricks

- [Use the automation controller CLI Tool](#)

- [Change the automation controller Admin Password](#)
- [Create an automation controller Admin from the commandline](#)
- [Set up a jump host to use with automation controller](#)
- [View Ansible outputs for JSON commands when using automation controller](#)
- [Locate and configure the Ansible configuration file](#)
- [View a listing of all ansible_ variables](#)
- [The ALLOW_JINJA_IN_EXTRA_VARS variable](#)
- [Configure the controllerhost hostname for notifications](#)
- [Launch Jobs with curl](#)
- [Filter instances returned by the dynamic inventory sources in automation controller](#)
- [Use an unreleased module from Ansible source with automation controller](#)
- [Connect to Windows with winrm](#)
- [Import existing inventory files and host/group vars into automation controller](#)

20.1. The automation controller CLI Tool

Automation controller has a full-featured command line interface.

For more information on configuration and use, see the [AWX Command Line Interface](#) and the [AWX manage utility](#) section.

20.2. Change the automation controller Administrator Password

During the installation process, you are prompted to enter an administrator password that is used for the `admin` superuser or system administrator created by automation controller. If you log in to the instance by using SSH, it tells you the default administrator password in the prompt.

If you need to change this password at any point, run the following command as root on the automation controller server:

```
awx-manage changepassword admin
```

Next, enter a new password. After that, the password you have entered works as the administrator password in the web UI.

To set policies at creation time for password validation using Django, see [Django password policies](#).

20.3. Create an automation controller Administrator from the command line

Occasionally you might find it helpful to create a system administrator (superuser) account from the command line.

To create a superuser, run the following command as root on the automation controller server and enter the administrator information as prompted:

```
awx-manage createsuperuser
```

20.4. Configuring automation controller to use jump hosts connecting to managed nodes

Credentials supplied by automation controller do not flow to the jump host through ProxyCommand. They are only used for the end-node when the tunneled connection is set up.

20.4.1. Configure a fixed user/keyfile in your SSH configuration file

You can configure a fixed user/keyfile in your SSH configuration file in the ProxyCommand definition that sets up the connection through the jump host.

Prerequisites

- Check whether all jump hosts are reachable from any node that establishes an SSH connection to the managed nodes, such as a Hybrid Controller or an Execution Node.

Procedure

1. Create an SSH configuration file `/var/lib/awx .ssh/config` on each node with the following details

```
Host jumphost.example.com
  Hostname jumphost.example.com
  User <jumphostuser>
  Port <jumphostport>
  IdentityFile ~/.ssh/id_rsa
  StrictHostKeyChecking no
  ProxyCommand ssh -W %h:%p jumphost.example.com
```

- The code specifies the configuration required to connect to the jump host 'jumphost.example.com'
- Automation controller establishes an SSH connection from each node to the managed nodes.
- Example values `jumphost.example.com`, `jumphostuser`, `jumphostport` and `~/.ssh/id_rsa` must be changed according to your environment
- Add a Host matching block to the already created SSH configuration file

`/var/lib/awx/.ssh/config` on the node, for example:

```
Host 192.0.*  
...
```

- The `Host 192.0.*` line indicates that all hosts in that subnet use the settings defined in that block. Specifically all hosts in that subnet are accessed using the `ProxyCommand` setting and connect through `jumphost.example.com`
- If `Host *` is used to indicate that all hosts connect through the specified proxy, ensure that `jumphost.example.com` is excluded from that matching, for example:

```
Host * !jumphost.example.com  
...
```

Using the Red Hat Ansible Automation Platform UI

2. On the navigation panel, select **Settings › Automation Execution › Job**
3. Click **[Edit]** and add `/var/lib/awx .ssh:/home/runner/.ssh:0` to the **Paths to expose isolated jobs** field.
4. Click **[Save]** to save your changes.

20.4.2. Configuring jump hosts using Ansible Inventory variables

You can add a jump host to your automation controller instance through Inventory variables.

These variables can be set at either the inventory, group, or host level. Use this method if you want to control the use of jump hosts inside automation controller using the inventory.

Procedure

- Go to your inventory and in the `variables` field of whichever level you choose, add the following variables:

```
ansible_user: <user_name>  
ansible_connection: ssh  
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q  
<user_name>@<jump_server_name>"'
```

20.5. View Ansible outputs for JSON commands when using automation controller

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

To view the Ansible outputs, browse to `https://<controller server`

name>/api/v2/jobs/<job_id>/job_events/

20.6. Locate and configure the Ansible configuration file

While Ansible does not require a configuration file, OS packages often include a default one in `/etc/ansible/ansible.cfg` for possible customization.

To use a custom `ansible.cfg` file, place it at the root of your project. Automation controller runs `ansible-playbook` from the root of the project directory, where it finds the custom `ansible.cfg` file.

NOTE An `ansible.cfg` file anywhere else in the project is ignored.

To learn which values you can use in this file, see [Generating a sample ansible.cfg file](#) in the Ansible documentation.

Using the defaults are acceptable for starting out, but you can configure the default module path or connection type here, as well as other things.

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

20.7. View a listing of all ansible_ variables

By default, Ansible gathers "facts" about the machines under its management, accessible in Playbooks and in templates.

To view all facts available about a machine, run the `setup` module as an *ad hoc* action:

```
ansible -m setup hostname
```

This prints out a dictionary of all facts available for that particular host. For more information, see [information-discovered-from-systems-facts](#) in the Ansible documentation.

20.8. The ALLOW_JINJA_IN_EXTRA_VARS variable

Setting `ALLOW_JINJA_IN_EXTRA_VARS = template` only works for saved job template extra variables.

Prompted variables and survey variables are excluded from the 'template'.

This parameter has three values:

- **Only On Template Definitions** to allow usage of Jinja saved directly on a job template definition (the default).
- **Never** to disable all Jinja usage (recommended).

- **Always** to always allow Jinja (strongly discouraged, but an option for prior compatibility).

This parameter is configurable in the **Jobs Settings** page of the automation controller UI.

20.9. Configuring the **controllerhost** hostname for notifications

From the **System settings** page, you can replace `https://controller.example.com` in the **Base URL of the Service** field with your preferred hostname to change the notification hostname.

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

20.10. Launching Jobs with curl

Launching jobs with the automation controller API is simple.

The following are some easy to follow examples using the **curl** tool.

Assuming that your Job Template ID is '1', your controller IP is 192.168.42.100, and that **admin** and **awxsecret** are valid login credentials, you can create a new job this way:

```
curl -f -k -H 'Content-Type: application/json' -XPOST \  
  --user admin:awxsecret \  
  https://192.168.42.100/api/v2/job_templates/1/launch/
```

This returns a JSON object that you can parse and use to extract the 'id' field, which is the ID of the newly created job. You can also pass extra variables to the Job Template call, as in the following example:

```
curl -f -k -H 'Content-Type: application/json' -XPOST \  
  -d '{"extra_vars": "{\"foo\": \"bar\"}"}' \  
  --user admin:awxsecret https://192.168.42.100/api/v2/job_templates/1/launch/
```

NOTE

The **extra_vars** parameter must be a string which contains JSON, not just a JSON dictionary. Use caution when escaping the quotes, etc.

20.11. Filtering instances returned by the dynamic inventory sources in the controller

By default, the dynamic inventory sources in automation controller (such as AWS and Google) return all instances available to the cloud credentials being used. They are automatically joined into groups based on various attributes. For example, AWS instances are grouped by region, by tag name, value, and security groups. To target specific instances in your environment, write your playbooks so that they target the generated group names.

For example:

```
---  
- hosts: tag_Name_webserver  
  tasks:  
  ...
```

You can also use the `Limit` field in the Job Template settings to limit a playbook run to a certain group, groups, hosts, or a combination of them. The syntax is the same as the `--limit parameter` on the `ansible-playbook` command line.

You can also create your own groups by copying the auto-generated groups into your custom groups. Make sure that the `Overwrite` option is disabled on your dynamic inventory source, otherwise subsequent synchronization operations delete and replace your custom groups.

20.12. Use an unreleased module from Ansible source with automation controller

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

First, determine which is the updated module you want to use from the available Ansible Core Modules or Ansible Extra Modules GitHub repositories.

Next, create a new directory, at the same directory level of your Ansible source playbooks, named `/library`.

When this is created, copy the module you want to use and drop it into the `/library` directory. It is consumed first by your system modules and can be removed once you have updated the stable version with your normal package manager.

20.13. Use callback plugins with automation controller

Ansible has a flexible method of handling actions during playbook runs, called callback plugins. You can use these plugins with automation controller to do things such as notify services upon playbook runs or failures, or send emails after every playbook run.

For official documentation on the callback plugin architecture, see [Developing plugins](#).

NOTE

Automation controller does not support the `stdout` callback plugin because Ansible only permits one, and it is already being used for streaming event data.

You might also want to review some example plugins, which should be modified for site-specific purposes, such as those available at: <https://github.com/ansible/ansible/tree/devel/lib/ansible/plugins/callback>

To use these plugins, put the callback plugin `.py` file into a directory called `/callback_plugins`

alongside your playbook in your automation controller Project. Then, specify their paths (one path per line) in the **Ansible Callback Plugins** field of the Job settings:



20 "win_user"
21]

Ansible Callback Plugins ⓘ Revert

1 []

Paths to expose to isolated jobs ⓘ Revert

1 []

Extra Environment Variables ⓘ Revert

1 { }

Save Revert all to default Cancel

NOTE

To have most callbacks shipped with Ansible applied globally, you must add them to the `callback_whitelist` section of your `ansible.cfg`.

If you have custom callbacks, see [Enabling callback plugins](#).


20.14. Connect to Windows with winrm

By default, automation controller attempts to `ssh` to hosts.

You must add the `winrm` connection information to the group variables to which the Windows hosts belong.

To get started, edit the Windows group in which the hosts reside and place the variables in the source or edit screen for the group.

To add `winrm` connection info:

- Edit the properties for the selected group by clicking on the Edit  icon of the group name that contains the Windows servers. In the "variables" section, add your connection information as follows: `ansible_connection: winrm`

When complete, save your edits. If Ansible was previously attempting an SSH connection and failed, you should re-run the job template.

20.15. Import existing inventory files and host/group vars into automation controller

To import an existing static inventory and the accompanying host and group variables into automation controller, your inventory must be in a structure similar to the following:

```
inventory/
```

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

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

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

If you only have a single flat file of inventory, a file called `ansible-hosts`, for example, import it as follows:

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

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

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

If you receive an error, such as:

```
ValueError: need more than 1 value to unpack
```

Create a directory to hold the hosts file, as well as the `group_vars`:

```
mkdir -p inventory-directory/group_vars
```

Then, for each of the groups that have `:vars` listed, create a file called `inventory-directory/group_vars/<groupname>` and format the variables in YAML format.

The importer then handles the conversion correctly.