Performance tuning for Ansible Automation Platform

This guide provides recommended practices for various processes needed to tune your Ansible Automation Platform deployments.

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. Types of workloads

This guide offers insights on enabling performance tuning for your Ansible Automation Platform deployment. It covers the following topics:

- The types of workloads and the components these workloads rely on to perform well
- The services each workload uses
- Reference workloads supported by the tested deployment models
- Reasons for scaling each service beyond the base configurations of the tested deployment models

1.1. UI authentication and platform load

Users can access the Ansible Automation Platform UI using enterprise authentication methods. The login performance with enterprise authentication is dependent on the performance of the external authentication provider and the complexity of the mapping of the external authentication attributes to RBAC attributes in Ansible Automation Platform. After successful login, the browser uses session authentication for subsequent requests, which is generally a faster authentication method as some session data is cached.

UI clients apply load to the platform gateway proxy and the gRPC authentication service, the web servers of all components, and the database, because most UI-driven API requests interact with the database.

The Ansible Automation Platform UI receives live updates through WebSockets and periodically requests updates from all components to properly render options and update on-screen information. Users can configure the frequency of these periodically maintained update requests by adjusting the "Refresh Interval" setting in the UI's **User Preferences**. This action affects the load that an open browser tab with the Ansible Automation Platform UI generates on backend services.

Through WebSockets in the user interface, the browser receives real-time job updates. The browser can subscribe to any job running on any automation controller node, because events are broadcasted and accessible everywhere. However, live streaming updates to the job detail page may increase the load on the automation controller service. To disable these updates, set `UI_LIVE_UPDATES_ENABLED` to false in the automation controller configuration.

NOTE	Disabling updates prevents the job detail page from automatically updating when events are received. In this case, you must manually refresh the page to access the most recent details.

1.2. REST API access and client load

The Red Hat Ansible Automation Platform provides a REST API, offering access to all its functionalities. This API can be accessed using various clients, including cURL, Python, Ansible Automation Platform configuration collections, or the Ansible URI module.

The API can be used to automate the following tasks:

- launching jobs
- updating inventories
- checking automation status
- pushing events into an Event Stream for Event-Driven Ansible
- automating the upload or publication of collections in automation hub
- managing automation execution environments in the automation hub container registry using a Podman client that connects to automation hub over its registry API

API clients apply load to the platform gateway proxy, the gRPC service for authentication, the web server of the targeted component, and the database, because most API client queries interact with the database.

To access the API, you can use the following common authentication methods: Basic authentication (using a username and password) and Token authentication (your chosen authentication method). We recommend Token authentication for better performance. You can create tokens through the platform gateway, which can be linked to an OAuth application or your account. The platform gateway's gRPC service authenticates each request and directs it to the appropriate application server based on the specified route. For more information, see Authenticating in the API.

1.3. Primary workloads for automation controller

The primary workloads for automation controller include the following:

- Managing automation content through automation controller projects
- Initiating automation by executing jobs

1.3.1. Automation controller project synchronization

Users define the source of automation content within the automation controller projects, such as Ansible Playbooks. The primary workload for these projects is synchronization, which is managed by project update jobs in the API. These jobs are also known as source control updates in the UI.

These project update jobs run exclusively on the control plane and in task pods within the OpenShift Container Platform. Their role is to update the automation controller with the latest automation content from its defined source, such as a Git repository.

Updating projects is not performance-sensitive, as long as they store only playbooks and Ansible-related text files. However, issues may arise when projects become excessively large. It is recommended to not store large volumes of binary data within a project. If jobs require access to additional data, it is recommended to retrieve this data from object storage or file storage within the playbook's scope.

1.3.2. Jobs and automation workloads

Jobs are the primary workload for automation controller and are run on the execution plane. They include the following job types:

- Standard jobs
- Workflow, sliced, and bulk jobs
- System jobs

Standard jobs

Standard jobs involve the execution of an Ansible Playbook from a Project against a set of hosts from an Inventory. Jobs are initiated by a control node, which then streams, processes, and stores job results.

A performance sensitive part of this is the processing of the playbook output. The output is captured and serialized into job events by the automation controller. A single Ansible task running against a host typically produces multiple job events (for example: task start, host-specific details, and task completion).

Event volume varies significantly with the playbook's configured verbosity level. For example, a simple debug task that prints `Hello World` on one host might produce 6 job events at verbosity 1, increasing to 34 job events at verbosity 3.

The dispatcher and the callback receiver collaborate to process, transmit, and store job events. These actions contribute to the platform's storage and processing overhead. Job events are processed on the control plane and stored in the database. The dispatcher processes job events, and the callback receiver stores them.

Workflow, Sliced and Bulk jobs

Standard jobs can be extended to enable more complex automation scenarios and orchestration by using the following job types:

- Sliced jobs: Split jobs to run against slices of the inventory in parallel
- Bulk jobs: Launch multiple jobs in a single request
- Workflow jobs: Coordinate multiple job templates

These job types coordinate the launch and management of multiple underlying standard jobs. They impact job scheduling, which occurs in the control plane, but otherwise do not have significant impact on their services.

System jobs

System jobs involve internal maintenance tasks, such as clean up of old job event data. The execution frequency of system jobs is managed by schedules. System jobs execute on the control plane, because they run management commands that interact with the database. These workloads involve key platform activities. Reducing the frequency of system jobs or increasing the number of days of data for jobs to retain can degrade database performance. In general, we recommend retaining as few days of data as possible and leveraging the external logging features for long term audit data storage. Storing more data in the database can make expensive queries that scan large tables more costly for the database.

1.4. Event-Driven Ansible activations

Activations are used by Event-Driven Ansible to run instances of `ansible-rulebook`. These activations can either connect to external event sources or listen to an event stream for incoming payloads.

Managing an activation and its output involves Event-Driven Ansible hybrid nodes, the platform gateway (when the events are sent to an event stream), the websocket server located within each API node or pods, and the database (where audit events are stored).

Activations process discrete payloads called events. The activation's resource usage is affected by the event arrival rate and the complexity of the rulebook's rules. When events match rules, they trigger actions, which launch jobs in automation controller. Event auditing stores audit events in the database and is enabled by default.

Each event is sent from the activation to the websocket server to be serialized and written to the database. This process stresses the server and can cause performance issues. Selecting **Skip audit events** in the UI for a given activation eliminates this workload. When **Skip audit events** is selected, rules are still fired, but the fire count in the API and UI is updated at a periodic interval (default 300 seconds) rather than immediately.

1.5. Collection synchronization

Private automation hub can synchronize collections from remote `ansible-galaxy` repositories, such as `galaxy.ansible.com` or automation hub on `console.redhat.com`.

Pulp content worker and the database are used to sync repositories. The automation controller can download these collections during project updates, or they can be used to build automation execution environments. Collections are also available for any other client using the `ansible-galaxy`

CLI to download and use.

The performance of syncing collections is relative to the number of collections listed in the `requirements.yml`, the number of versions synced, and the number of versions retained. Specifically, synchronization uses memory in proportion to the number of collections and versions synchronized. Using a targeted `requirements.yml` with specific versions can limit this impact. Hosting collections utilizes storage space, which can be managed by specifying the retained number of versions on the repository.

1.6. Container image hosting and performance

Private automation hub has the capability to host container images for automation execution environments and decision environments. These container images can be configured to be pulled by Event-Driven Ansible or automation controller when running activations or jobs. The frequency at which these containers are pulled is determined by the frequency of job starts and the pull policy configured for the automation execution environments and decision environments in Event-Driven Ansible or automation controller.

The performance of pushing and pulling container images from automation hub depends on the disk performance of the underlying storage, because Pulp content workers store and fetch the layers of the container image from disk. The size of layers can impact the memory used by the pulp workers, because they serve entire layers at once. The frequency of container image pulls is determined by the pull policy on jobs and activations, the frequency of job or activation starts, and whether these jobs or activations starts occur on nodes or Container Groups where the image has not yet been pulled.

1.7. Reference workloads for growth topologies

The following table provides reference data for typical workloads, performance metrics, and capacity planning for the tested Ansible Automation Platform growth topologies.

Table 1. Reference workloads for growth topologies

Component / Feature	**Metric**
REST API	8 requests per second (RPS)
REST API 50 percentile latency at 8 RPS	500 milliseconds
REST API 99 percentile latency at 8 RPS	1.5 seconds
Hosts in automation controller inventory	1,000 hosts
Job start rate in automation controller (max burst rate with standard launch)	20 jobs started per second
Concurrent jobs in automation controller	10 concurrent jobs with default forks (5 forks is default) 100 with forks=1

Component / Feature	Metric
Callback receiver event processing rate	10,000 job events per second at peak
Job History with 30 days retention	2kb event; 500 events per playbook run; 500 jobs a day Less than 60Gb (as specified as minimum required disk on Database node)
(Certified) Sync time	Less than 30 minutes
(Validated) Sync time	Less than 5 minutes
Activation processing events with skip audit events enabled (6 activation) with events incoming via Event Stream and execution strategy set to sequential (default) in the rulebook	1 actionable event/minute with minimal payload with job template action on local automation controller where each job completes in 1 minute

1.8. Reference workloads for enterprise topologies

The following table provides reference data for typical workloads, performance metrics, and capacity planning for the tested Ansible Automation Platform enterprise topologies.

Table 2. Reference workloads for enterprise topologies

Component / Feature	Metric
REST API	16 requests per second (RPS)
REST API 50 percentile latency at 16 RPS	500 milliseconds
REST API 99 percentile latency at 16 RPS	1.5 seconds
Hosts in automation controller inventory	10,000 hosts
Job start rate in automation controller	80 jobs started per second
Concurrent jobs in automation controller	40 with default forks (5 forks is default) 400 with forks=1
Callback receiver event rate	40,000 events per second at peak
Job History with 7 days retention	2kb event; 500 events per playbook run; 2000 jobs a day Less than 60Gb (as specified as minimum required disk on Database node)
(Certified) Sync time	Less than 30 minutes
(Validated) Sync time	Less than 5 minutes

Component / Feature	Metric
Processing events with skip audit events enabled (6 activations) with events incoming via Event Stream and execution strategy set to sequential (default) in the rulebook	3 actionable event/minute with minimal payload with job template action on local automation controller where each job completes in 1 minute