Red Hat

Optimize

Ansible Automation Platform 2.6

May 12, 2026

Contents

1 Optimize

Optimize platform performance

Optimizing platform performance ensures your deployment responds efficiently to workload demands and maintains availability under load. Scale services appropriately and tune configurations to support your automation capacity requirements.

Optimizing platform performance helps you to:

- **Maintain system responsiveness**: Scale services and adjust capacity to handle workload demands and prevent request queueing or timeouts.
- **Support growing automation workloads**: Plan deployment capacity based on your workload characteristics and add resources to accommodate increased managed hosts or concurrent jobs.
- **Resolve performance bottlenecks**: Monitor key performance indicators to identify services that require scaling and adjust configurations to improve throughput.

How Ansible Automation Platform supports performance optimization

You can use multiple optimization approaches based on your deployment type and workload characteristics:

- **Scaling strategies** include horizontal scaling (adding more replicas) and vertical scaling (adding CPU, memory, or other resources). Scale services for platform gateway, automation controller, Event-Driven Ansible, and automation hub independently based on performance indicators.
- **Database tuning** includes configuring PostgreSQL parameters for memory allocation and maintenance operations to improve query performance and data management.
- **Capacity planning** involves characterizing workload based on managed hosts, concurrent jobs, and API request rates, then planning control and execution capacity.

Automation execution settings

You can configure the following automation execution settings through the UI, API, or file settings:

- Live events in the automation controller UI
- Job event processing and scheduling

- Control and execution node capacity
- Instance and container groups capacity
- Internal cluster routing
- Web server tuning

Workload types and access methods

Learn about enabling performance tuning for your Ansible Automation Platform deployment.

This section 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

UI authentication and platform load

Users can access the Ansible Automation Platform UI via enterprise authentication. UI clients apply load to the platform gateway proxy, gRPC authentication service, component web servers, and the database, as most UI-driven API requests interact directly with the database.

Login performance with enterprise authentication depends on two factors: the performance of the external authentication provider, and the complexity of mapping external attributes to RBAC attributes in Ansible Automation Platform. After a successful login, the browser uses session authentication for subsequent requests. This method is generally faster because some session data is cached.

The Ansible Automation Platform UI receives live updates through WebSockets and periodically requests updates from all components. This is necessary to properly render options and update on-screen information.

Users can configure the frequency of these periodic update requests. This is done by adjusting the "Refresh Interval" setting in the UI User Preferences tab. 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 accessible everywhere.

However, live streaming updates to the job detail page might 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.

REST API access and client load

Ansible Automation Platform provides a REST API, offering access to all its functionalities. You can access this API by using various clients, including cURL, Python, Ansible Automation Platform configuration collections, or the Ansible URI module.

You can use the API 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). Token authentication is recommended for better performance.

Use the platform gateway to create tokens and link them 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.

Related information
Authenticating in the API

Vertically scale tested deployment models to improve performance

Vertical scaling increases the physical resources available to a service, including the CPU, memory, disk volume, and disk Input/Output Operations per Second (IOPS). Use vertical scaling for deployments with high resource utilization or workload demand.

Benefits of vertical scaling

- Relieves resource contention: Applications have access to more resources and this can relieve resource contention or exhaustion.

Drawbacks and other considerations for vertical scaling

- Extensive testing required: The installation program attempts to tune application and system configurations to use additional resources, but not all components of the application automatically scale in relation to machine size. Manually tuning each variable requires extensive testing. For this reason, after an instance size has been verified for an environment, horizontal scaling by adding more instances of the same size is recommended.
- Application-level limitations: For VM-based installation or Containerized deployments, instances with more than 64 CPU cores and 128 GB of RAM might not scale linearly due to system and application-level limits.
- Resource overcommit: Overcommitting virtual machine resources (for example, allocating more virtual CPU/RAM to guests than is physically available on the host) leads to unpredictable performance.
- CPU throttling: In OpenShift Container Platform, setting a CPU `limit` without an equivalent `request` can lead to CPU throttling, even if the node has spare CPU capacity. This throttling negatively impacts API latency.
    - To mitigate this, always set CPU `requests` equal to CPU `limits`.
    - Monitor CPU throttling using the `container_cpu_cfs_throttled_seconds_total` metric.
- Database limitations: Scaling the application increases the maximum potential number of database connections from worker processes and the overall memory, I/O, and CPU utilization of the PostgreSQL instance. As you scale past the tested deployment models, deploy your separate Postgres instances per component (platform gateway, Event-Driven Ansible, automation controller, automation hub).

Horizontally scale tested deployment models to improve performance

Horizontal scaling involves increasing the number of replicas (pods or virtual machines) for a given service. Similar to vertical scaling, this approach is useful for high resource utilization or workload scaling.

Benefits of horizontal scaling

- Improved availability: Distributes load across more instances to reduce the impact of a single slow or failing node.
- Redundancy: Provides extra capacity, allowing individual service nodes to recover or cool-off without impacting overall availability.
- Increased authentication capacity: Scaling the platform gateway directly increases the platform's authentication throughput, because each platform gateway pod includes its own authentication service.
- Repeatable scaling procedure: After the instance size and configuration are verified for your environment, deploy identical instances to scale.

Drawbacks and other considerations for horizontal scaling

- Database limitations: Scaling the application increases the maximum potential number of database connections from worker processes. This also increases the overall memory, I/O, and CPU utilization of the PostgreSQL instance. When you scale past the tested deployment models, deploy separate PostgreSQL instances per component (platform gateway, Event-Driven Ansible, automation controller, automation hub).
- Health Check Overhead: In a mesh architecture, each Envoy proxy sends health checks to every other cluster member. Horizontal scaling increases this baseline traffic, adding to system usage.

Horizontally scale in Event-Driven Ansible

You can set up multi-node deployments for components across Ansible Automation Platform. Whether you require horizontal scaling for Automation Execution, Automation Decisions, or automation mesh, you can scale your deployments based on your organization's needs.

Horizontal scaling in Event-Driven Ansible controller

With Event-Driven Ansible controller, you can set up horizontal scaling for events automation. This multi-node deployment allows you to define as many nodes as preferred during installation. You can also increase or decrease the node count at any time to meet organizational needs.

The following node types are used in this deployment:

API node type

Responds to the HTTP REST API of Event-Driven Ansible controller.

Worker node type

Runs an Event-Driven Ansible worker, which is the component of Event-Driven Ansible that not only manages projects and activations, but also executes the activations themselves.

Hybrid node type

Is a combination of the API node and the worker node.

Inventory file updates for RPM-based installations (VMs)

The following example shows how you can set up an inventory file for horizontal scaling of Event-Driven Ansible controller on Red Hat Enterprise Linux VMs using the host group name `[automationedacontroller]` and the node type variable `eda_node_type`:

```
[automationedacontroller]


3.88.116.111 routable_hostname=automationedacontroller-api.example.com
eda_node_type=api


# worker node

3.88.116.112 routable_hostname=automationedacontroller-api.example.com
eda_node_type=worker
```

Inventory file updates for containerized installations

The following example shows how you can set up an inventory file for horizontal scaling of Event-Driven Ansible controller on Red Hat Enterprise Linux VMs using the host group name `[automationeda]` and the node type variable `eda_type`:

```
[automationeda]


3.88.116.111 routable_hostname=automationeda-api.example.com eda_type=api


# worker node
3.88.116.112 routable_hostname=automationeda-api.example.com eda_type=worker
```

Sizing and scaling guidelines

Scaling guidelines for Event-Driven Ansible define how to properly size API nodes based on user requests and worker nodes based on automation activations. You can ensure independent scaling and optimized resource utilization for your environment by separating the node roles.

API nodes process user requests (interactions with the UI or API) while worker nodes process the activations and other background tasks required for Event-Driven Ansible to function properly. The number of API nodes you require correlates to the required number of users of the application and the number of worker nodes correlates to the required number of activations you want to run.

Since activations are variable and controlled by worker nodes, the supported approach for scaling is to use separate API and worker nodes instead of hybrid nodes due to the efficient allocation of hardware resources by worker nodes. By separating the nodes, you can scale each type independently based on specific needs, leading to better resource utilization and cost efficiency.

An example of an instance in which you might consider scaling up your node deployment is when you want to deploy Event-Driven Ansible for a small group of users who will run a large number of activations. In this case, one API node is adequate, but if you require more, you can scale up to three additional worker nodes.

Set up horizontal scaling for Event-Driven Ansible controller

To scale up (add more nodes) or scale down (remove nodes), you must update the content of the inventory file to add or remove nodes and rerun the installation program.

Procedure

1. Update the inventory to add two more worker nodes:
   - For RPM-based installations:

```
[automationedacontroller]


3.88.116.111 routable_hostname=automationedacontroller-api.example.com
eda_node_type=api


3.88.116.112 routable_hostname=automationedacontroller-api.example.com
eda_node_type=worker


# two more worker nodes

3.88.116.113 routable_hostname=automationedacontroller-api.example.com
eda_node_type=worker


3.88.116.114 routable_hostname=automationedacontroller-api.example.com
eda_node_type=worker
```

   - For containerized installations:

```
[automationeda]


3.88.116.111 routable_hostname=automationeda-api.example.com eda_type=api


3.88.116.112 routable_hostname=automationeda-api.example.com
eda_type=worker


# two more worker nodes

3.88.116.113 routable_hostname=automationeda-api.example.com
eda_type=worker


3.88.116.114 routable_hostname=automationeda-api.example.com
eda_type=worker
```

2. Re-run the installer.

Scale and manage by installation type

Considerations for scaling and managing each Ansible Automation Platform component differ based on the installation type. The following table provides information on scaling and management operations for each installation type, and other common operations to consider when planning your deployment:

NOTE:
Red Hat OpenShift Container Platform-based deployments provide the most flexibility and customizability, which enables you to adapt the deployment as needs change. OpenShift Container Platform also provides fine-grained observability through its built-in metrics capabilities and integrations, with log aggregation tools that capture all logs from all pods in the Ansible Automation Platform deployment.

Comparison of Scaling and Management Operations by installation Type

Task	OpenShift Container Platform	VM-based installation	Containerized Deployments (Podman based)
Horizontally sca	Scale control, execution, automation hub, and Event-Driven Ansible components independently by adjusting replicas.	Requires updating inventory file and re-running entire installation, which restarts and requires	Requires updating inventory file and re-running entire installation, which restarts and requires

Task	OpenShift Container Platform	VM-based installation	Containerized Deployments (Podman based)
le up	Expanding total capacity by adding worker nodes to OpenShift Container Platform does not disrupt Ansible Automation Platform operation.	halting use of the platform.	halting use of the platform.
Ho riz ont ally sca le do wn	Reduces replicas handled by operator. For scaling down automation controller task pods, usage of `termination_grace_period_seconds` allows the scale down to occur after jobs are drained from the task pod.	Requires updating the inventory file and re-running the entire installation. This restarts the platform, halting use.	Requires updating the inventory file and re-running the entire installation. This restarts the platform, halting use.
Ve rtic ally sca lin g up or do wn	Increases or decreases requests and limits on individual deployment types. The operator deploys a new pod with these resources, and previous pods scale down. For automation controller task pods, usage of `termination_grace_period_seconds allows` the old replicas to scale down after jobs are drained from task pods.	Depending on your virtualization provider, the virtual machine might require shutdown to resize. Attaining the full benefit of vertical scaling requires re-running the installation program, which restarts and halts use of the platform. Running the installation program adapts any settings that were tuned based on the number of available cores and RAM.	Depending on your virtualization provider, the virtual machine might require shutdown to resize. Attaining the full benefit of vertical scaling requires re-running the installation program, which restarts and halts use of the platform. Running the installation program adapts any settings that were tuned based on the number of available cores and RAM.
Ins tall ati on	Utilizes OpenShift Container Platform Operators for automated deployment and management.	Ansible Playbook-based installation program installs RPMs and configures the platform.	Ansible Playbook-based installation program that configures platform services in Podman containers, which are managed by `systemd`.
Up gra de	Handled by OpenShift Container Platform Operators with automated rolling updates. Usage of `termination_grace_period_seconds` allows	Requires running the installation program and restarting services, which halts use of the platform.	Requires running the installation program and restarting services, which halts use of the platform.

Task	OpenShift Container Platform	VM-based installation	Containerized Deployments (Podman based)
	upgrades without downtime and without halting job execution.		
Aggregating Application Logs	Centralized logging through OpenShift Container Platform's built-in logging stack or integrations with external aggregators.	Requires external log aggregation solutions (e.g., ELK stack, Splunk) to collect logs from individual nodes.	Requires external log aggregation solutions (e.g., ELK stack, Splunk) to collect logs from container instances.
Monitoring Resource Utilization	Comprehensive monitoring with OpenShift Container Platform's built-in Prometheus and Grafana dashboards, providing detailed pod and node metrics.	Requires external monitoring tools to collect and visualize resource metrics from nodes.	Requires external monitoring tools to collect and visualize resource metrics from nodes.

Performance tuning for operator environments

A pod in Kubernetes is the smallest deployable compute unit, consisting of one or more containers sharing networking and storage on a single host. Red Hat Ansible Automation Platform uses a default pod specification, which can be customized with a user-defined YAML or JSON document.

Introduction

The Kubernetes concept of a pod is one or more containers deployed together on one host, and the smallest compute unit that can be defined, deployed, or managed.

Pods are the equivalent of a machine instance (physical or virtual) to a container. Each pod is allocated its own internal IP address, therefore owning its entire port space, and containers within pods can share their local storage and networking.

Pods have a life cycle. They are defined, then they are assigned to run on a node, then they run until their containers exit or they are removed for some other reason. Pods, depending on policy and exit code, can be removed after exiting, or can be retained to enable access to the logs of their containers.

Red Hat Ansible Automation Platform provides a simple default pod specification, however, you can provide a custom YAML, or JSON document that overrides the default pod specification. This custom document uses custom fields, such as `ImagePullSecrets`, that can be serialized as valid Pod JSON or YAML.

Example of a pod that provides a long-running service

This example demonstrates many features of pods, most of which are discussed in other topics and thus only briefly mentioned here:

```
apiVersion: v1
kind: Pod
metadata:
  annotations: { ... }
  labels:
    deployment: docker-registry-1
    deploymentconfig: docker-registry
    docker-registry: default
  generateName: docker-registry-1-
spec:
  containers:
  - env:
    - name: OPENSHIFT_CA_DATA
      value: ...
    - name: OPENSHIFT_CERT_DATA
      value: ...
    - name: OPENSHIFT_INSECURE
      value: "false"
    - name: OPENSHIFT_KEY_DATA
      value: ...
    - name: OPENSHIFT_MASTER
      value: https://master.example.com:8443
    image: openshift/origin-docker-registry:v0.6.2
    imagePullPolicy: IfNotPresent
    name: registry
    ports:
```

```
    - containerPort: 5000
      protocol: TCP
    resources: {}
    securityContext: { ... }              volumeMounts:
    - mountPath: /registry
      name: registry-storage
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: default-token-br6yz
      readOnly: true
  dnsPolicy: ClusterFirst
  imagePullSecrets:
  - name: default-dockercfg-at06w
  restartPolicy: Always
  serviceAccount: default
  volumes:
  - emptyDir: {}
    name: registry-storage
  - name: default-token-br6yz
    secret:
      secretName: default-token-br6yz
```

Label	Description
`annotations:`	Pods can be "tagged" with one or more labels, which can then be used to select and manage groups of pods in a single operation. The labels are stored in key:value format in the metadata hash. One label in this example is `docker-registry=default`.
`generateName:`	Pods must have a unique name within their namespace. A pod definition can specify the basis of a name with the `generateName` attribute, and add random characters automatically to generate a unique name.
`containers:`	`containers` specifies an array of container definitions. In this case (as with most), defines only one container.
`env:`	Environment variables pass necessary values to each container.
`image:`	Each container in the pod is instantiated from its own Docker-formatted container image.

Label	Description
`ports:`	The container can bind to ports made available on the pod's IP.
`resources:`	When you specify a pod, you can optionally describe how much of each resource a container needs. The most common resources to specify are CPU and memory (RAM). Other resources are available.
`securityContext:`	OpenShift Online defines a security context for containers that specifies whether they are permitted to run as privileged containers, run as a user of their choice, and more. The default context is very restrictive but administrators can change this as required.
`volumeMounts:`	The container specifies where external storage volumes should be mounted within the container. In this case, there is a volume for storing the registry's data, and one for access to credentials the registry needs for making requests against the OpenShift Online API.
`ImagePullSecrets`	A pod can contain one or more containers, which must be pulled from some registry. If containers come from registries that require authentication, you can give a list of `ImagePullSecrets:` that refer to `ImagePullSecrets` present in the namespace. Having these specified enables Red Hat OpenShift Container Platform to authenticate with the container registry when pulling the image.
`restartPolicy:`	The pod restart policy with possible values `Always`, `OnFailure`, and `Never`. The default value is `Always`.
`serviceAccount:`	Pods making requests against the OpenShift Online API is a common enough pattern that there is a `serviceAccount` field for specifying which service account user the pod should authenticate as when making the requests. This enables fine-grained access control for custom infrastructure components.
`volumes:`	The pod defines storage volumes that are available to its container(s) to use. In this case, it provides an ephemeral volume for the registry storage and a secret volume containing the service account credentials.

You can change the pod used to run jobs in a Kubernetes-based cluster by using automation controller and editing the pod specification in the automation controller UI. The pod specification that is used to create the pod that runs the job is in YAML format.

Related information
OpenShift Online
Resource Management for Pods and Containers

Customizing the pod specification

Customize pod specifications to improve performance

You can use the following procedure to customize the pod.

Procedure

1. In the navigation panel, select **Automation Execution > Infrastructure > Instance Groups**.
2. Check **Customize pod specification**.
3. In the **Pod Spec Override** field, specify the namespace by using the toggle to enable and expand the **Pod Spec Override** field.
4. Click **Save**.
5. Optional: Click **Expand** to view the entire customization window if you want to provide additional customizations.

Next steps

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

Enable pods to reference images from other secured registries

If a container group uses a container from a secured registry that requires a credential, you can associate a Container Registry credential with the Execution Environment that is assigned to the job template.

Automation controller uses this to create an `ImagePullSecret` for you in the OpenShift Container Platform namespace where the container group job runs, and cleans it up after the job is done.

Alternatively, if the `ImagePullSecret` already exists in the container group namespace, you can specify the `ImagePullSecret` in the custom pod specification for the `ContainerGroup`.

Note that the image used by a job running in a container group is always overridden by the Execution Environment associated with the job.

Use of pre-created ImagePullSecrets (Advanced) If you want to use this workflow and pre-create the `ImagePullSecret`, you can source the necessary information to create it from your local `.dockercfg` file on a system that has previously accessed a secure container registry.

The `.dockercfg file`, or `$HOME/.docker/config.json` for newer Docker clients, is a Docker credentials file that stores your information if you have previously logged into a secured or insecure registry.

Procedure

1. If you already have a `.dockercfg` file for the secured registry, you can create a secret from that file by running the following command:

```
$ oc create secret generic <pull_secret_name> \
--from-file=.dockercfg=<path/to/.dockercfg> \
--type=kubernetes.io/dockercfg
```

2. Or if you have a `$HOME/.docker/config.json` file:

```
$ oc create secret generic <pull_secret_name> \
--from-file=.dockerconfigjson=<path/to/.docker/config.json> \
--type=kubernetes.io/dockerconfigjson
```

3. If you do not already have a Docker credentials file for the secured registry, you can create a secret by running the following command:

```
$ oc create secret docker-registry <pull_secret_name> \
--docker-server=<registry_server> \
--docker-username=<user_name> \
--docker-password=<password> \
--docker-email=<email>
```

4. To use a secret for pulling images for pods, you must add the secret to your service account. The name of the service account in this example must match the name of the service account the pod uses. The default is the default service account.

```
$ oc secrets link default <pull_secret_name> --for=pull
```

5. Optional: To use a secret for pushing and pulling build images, the secret must be mountable inside a pod. You can do this by running:

```
$ oc secrets link builder <pull_secret_name>
```

6. Optional: For builds, you must also reference the secret as the pull secret from within your build configuration.

Result

When the container group is successfully created, the **Details** tab of the newly created container group remains. This allows you to review and edit your container group information. This is the same menu that is opened if you click the **Edit** icon ✎ from the **Instance Group** link. You can also edit instances and review jobs associated with this instance group.

Manage resources for pods and containers

When you specify a pod, you can specify how much of each resource a container needs. The most common resources to specify are CPU and memory (RAM).

When you specify the resource request for containers in a Pod, the Kubernetes-scheduler uses this information to allocate the node to place the Pod on.

When you specify a resource limit for a container, the *kubelet*, or node agent, enforces those limits so that the running container is not permitted to use more of that resource than the limit you set. The kubelet also reserves at least the requested amount of that system resource specifically for that container to use.

Requests and limits

If the node where a pod is running has enough resources available, it is possible for a container to use more resources than its request for that resource specifies. However, a container is not allowed to use more than its resource limit.

For example, if you set a memory request of 256 MiB for a container, and that container is in a pod scheduled to a Node with 8GiB of memory and no other pods, then the container can try to use more RAM.

If you set a memory limit of 4GiB for that container, the kubelet and container runtime enforce the limit. The runtime prevents the container from using more than the configured resource limit.

If a process in the container tries to consume more than the allowed amount of memory, the system kernel terminates the process that attempted the allocation, with an *Out Of Memory* (OOM) error.

You can implement limits in two ways:

- Reactively: the system intervenes once it sees a violation.
- By enforcement: the system prevents the container from ever exceeding the limit.

Different runtimes can have different ways to implement the same restrictions.

NOTE:
If you specify a limit for a resource, but do not specify any request, and no admission-time mechanism has applied a default request for that resource, then Kubernetes copies the limit you specified and uses it as the requested value for the resource.

Resource types

CPU and memory are both resource types. A resource type has a base unit. CPU represents compute processing and is specified in units of Kubernetes CPUs. Memory is specified in units of bytes.

CPU and memory are collectively referred to as compute resources, or resources. Compute resources are measurable quantities that can be requested, allocated, and consumed. They are distinct from API resources. API resources, such as pods and services, are objects that can be read and modified through the Kubernetes API server.

Specify resource requests and limits for pods and containers

For each container, you can specify resource limits and requests, including the following:

```
spec.containers[].resources.limits.cpu
spec.containers[].resources.limits.memory
spec.containers[].resources.requests.cpu
spec.containers[].resources.requests.memory
```

Although you can only specify requests and limits for individual containers, it is also useful to think about the overall resource requests and limits for a pod. For a particular resource, a pod resource request or limit is the sum of the resource requests or limits of that type for each container in the pod.

Resource units in Kubernetes

CPU resource units Limits and requests for CPU resources are measured in CPU units. In Kubernetes, one CPU unit is equal to one physical processor core, or one virtual core, depending on whether the node is a physical host or a virtual machine running inside a physical machine.

Fractional requests are allowed. When you define a container with `spec.containers[].resources.requests.cpu` set to `0.5`, you are requesting half as much CPU time compared to if you asked for 1.0 CPU. For CPU resource units, the quantity expression 0.1 is equivalent to the expression 100m, which can be read as one hundred millicpu or one hundred millicores. millicpu and millicores mean the same thing. CPU resource is always specified as an absolute amount of resource, never as a relative amount. For example, 500m CPU represents the same amount of computing power whether that container runs on a single-core, dual-core, or 48-core machine.

NOTE:
To specify CPU units less than 1.0 or 1000m you must use the milliCPU form. For example, use 5m, not 0.005 CPU.

Memory resource units Limits and requests for memory are measured in bytes. You can express memory as a plain integer or as a fixed-point number using one of these quantity suffixes: E, P, T, G, M, k. You can also use the power-of-two equivalents: Ei, Pi, Ti, Gi, Mi, Ki. For example, the following represent roughly the same value:

```
128974848, 129e6, 129M,  128974848000m, 123Mi
```

Pay attention to the case of the suffixes. If you request 400m of memory, this is a request for 0.4 bytes, not 400 mebibytes (400Mi) or 400 megabytes (400M).

Example CPU and memory specification The following cluster has enough free resources to schedule a task pod with a dedicated 100m CPU and 250Mi. The cluster can also withstand bursts over that dedicated usage up to 2000m CPU and 2Gi memory.

```
spec:
  task_resource_requirements:
    requests:
      cpu: 100m
      memory: 250Mi
    limits:
      cpu: 2000m
      memory: 2Gi
```

Automation controller will not schedule jobs that use more resources than the limit set. If the task pod does use more resources than the limit set, the container is OOMKilled by Kubernetes and restarted.

Size recommendations for resource requests

All jobs that use a container group use the same pod specification. The pod specification includes the resource requests for the pod that runs the job.

All jobs use the same resource requests. The specified resource requests for your particular job on the pod specification affect how Kubernetes schedules the job pod based on resources available on worker nodes. These are the default values.

- One fork typically requires 100Mb of memory. This is set by using `system_task_forks_mem`. If your jobs have five forks, the job pod specification must request 500Mb of memory.
- For job templates that have a particularly high forks value or otherwise need larger resource requests, you should create a separate container group with a different pod spec that indicates larger resource requests. Then you can assign it to the job template. For example, a job template with the forks value of 50 must be paired with a container group that requests 5GB of memory.
- If the fork value for a job is high enough that no single pod would be able to contain the job, use the job slicing feature. This splits the inventory up such that the individual job "slices" fit in an automation pod provisioned by the container group.

Adjust the control plane to tune performance

The control plane refers to the automation controller pods which contain the web and task containers that, among other things, provide the user interface and handle the scheduling and launching of jobs.

On the automation controller custom resource, the number of *replicas* determines the number of automation controller pods in the automation controller deployment.

Requests and limits for task containers

You must set a value for the task container's CPU and memory resource limits. For each job that is run in an execution node, processing must occur on the control plane to schedule, open, and receive callback events for that job.

For Operator deployments of automation controller, this control plane capacity usage is tracked on the **controlplane** instance group. The available capacity is determined based on the limits the user sets on the task container, using the **task_resource_requirements** field in the automation controller specification, or in the OpenShift UI, when creating automation controller.

You can also set memory and CPU resource limits that make sense for your cluster.

Containers resource requirements

You can configure the resource requirements of tasks and the web containers, at both the lower end (requests) and the upper end (limits). The execution environment control plane is used for project updates, but is normally the same as the default execution environment for jobs.

Setting resource requests and limits is a best practice because a container that has both defined is given a higher *Quality of Service* class. This means that if the underlying node is resource constrained and the cluster has to reap a pod to prevent running memory or other failure, the control plane pod is less likely to be reaped.

These requests and limits apply to the control pods for automation controller and if limits are set, determine the *capacity* of the instance. By default, controlling a job takes one unit of capacity. The memory and CPU limits of the task container are used to determine the capacity of control nodes. For more information about how this is calculated, see Automation controller capacity determination and job impact.

See also Jobs scheduled on the worker nodes.

Name	Description	Default
`web_resource_requirements`	Web container resource requirements	requests: {CPU: 100m, memory: 128Mi}
`task_resource_requirements`	Task container resource requirements	requests: {CPU: 100m, memory: 128Mi}
`ee_resource_requirements`	EE control plane container resource requirements	requests: {CPU: 100m, memory: 128Mi}
`redis_resource_requirements`	Redis control plane container resource requirements	requests: {CPU:100m, memory: 128Mi}

The use of `topology_spread_constraints` to maximally spread control nodes onto separate underlying Kubernetes worker nodes is recommended. A reasonable set of requests and limits would be limits whose sum is equal to the actual resources on the node. If only `limits` are set, then the request is automatically set to be equal to the limit. But because some variability of resource usage between the containers in the control pod is permitted, you can set `requests` to a lower amount, for example to 25% of the resources available on the node. An example of container customization for a cluster where the worker nodes have 4 CPUs and 16 GB of RAM could be:

```
spec:
  ...
  web_resource_requirements:
    requests:
      cpu: 250m
      memory: 1Gi
    limits:
      cpu: 1000m
      memory: 4Gi
  task_resource_requirements:
    requests:
      cpu: 250m
      memory: 1Gi
    limits:
      cpu: 2000m
      memory: 4Gi
  redis_resource_requirements
    requests:
      cpu: 250m
      memory: 1Gi
    limits:
      cpu: 1000m
      memory: 4Gi
  ee_resource_requirements:
    requests:
      cpu: 250m
      memory: 1Gi
    limits:
      cpu: 1000m
      memory: 4Gi
```

Alternative capacity limiting with automation controller settings

The capacity of a control node in OpenShift is determined by the memory and CPU limits. However, if these are not set then the capacity is determined by the memory and CPU detected by the pod on the filesystem, which are actually the CPU and memory of the underlying Kubernetes node.

This can lead to issues with overwhelming the underlying Kubernetes pod if the automation controller pod is not the only pod on that node. If you do not want to set limits directly on the task container, you can use `extra_settings`.

```
SYSTEM_TASK_ABS_MEM = 3gi

SYSTEM_TASK_ABS_CPU = 750m
```

This acts as a soft limit within the application that enables automation controller to control how much work it attempts to run, while not risking any CPU throttling from Kubernetes itself, or being reaped if memory usage peaks above the required limit. These settings accept the same format accepted by resource requests and limits in the Kubernetes resource definition.

Related information
Specify nodes for job execution

Specify dedicated nodes for pods and job execution

A Kubernetes cluster runs on multiple nodes. Use the `topologySpreadConstraints` setting to control how pods are distributed across your nodes during scheduling. This ensures high availability and balanced workloads across your infrastructure.

Do not schedule your pods on a single node, because if that node fails, the services that those pods provide also fails.

Schedule the control plane nodes to run on different nodes to the automation job pods. If the control plane pods share nodes with the job pods, the control plane can become resource starved and degrade the performance of the whole application.

Assign pods to specific nodes

You can constrain the automation controller pods created by the operator to run on a certain subset of nodes.

- `node_selector` and `postgres_selector` constrain the automation controller pods to run only on the nodes that match all the specified key, or value, pairs.
- `tolerations` and `postgres_tolerations` enable the automation controller pods to be scheduled onto nodes with matching taints. See Taints and Toleration in the Kubernetes documentation for further details.

The following table shows the settings and fields that can be set on the automation controller's specification section of the YAML (or using the OpenShift UI form).

Name	Description	Default
`postgres_image`	Path of the image to pull	postgres
`postgres_image_version`	Image version to pull	13
`node_selector`	AutomationController pods' nodeSelector	""
`topology_spread_constraints`	AutomationController pods' topologySpreadConstraints	""
`tolerations`	AutomationController pods' tolerations	""
`annotations`	AutomationController pods' annotations	""
`postgres_selector`	Postgres pods' nodeSelector	""
`postgres_tolerations`	Postgres pods' tolerations	""

`topology_spread_constraints` can help optimize spreading your control plane pods across the compute nodes that match your node selector. For example, with the `maxSkew` parameter of this option set to `100`, this means maximally spread across available nodes. So if there are three matching compute nodes and three pods, one pod will be assigned to each compute node. This parameter helps prevent the control plane pods from competing for resources with each other.

Example of a custom configuration for constraining controller pods to specific nodes

```
spec:
  ...
  node_selector: |
    disktype: ssd
    kubernetes.io/arch: amd64
    kubernetes.io/os: linux
  topology_spread_constraints: |
    - maxSkew: 100
      topologyKey: "topology.kubernetes.io/zone"
      whenUnsatisfiable: "ScheduleAnyway"
      labelSelector:
        matchLabels:
          app.kubernetes.io/name: "<resourcename>"
  tolerations: |
    - key: "dedicated"
      operator: "Equal"
      value: "AutomationController"
      effect: "NoSchedule"
  postgres_selector: |
    disktype: ssd
    kubernetes.io/arch: amd64
    kubernetes.io/os: linux
  postgres_tolerations: |
    - key: "dedicated"
      operator: "Equal"
      value: "AutomationController"
      effect: "NoSchedule"
```

Specify nodes for job execution

You can add a node selector to the container group pod specification to ensure they only run against certain nodes. First add a label to the nodes you want to run jobs against.

The following procedure adds a label to a node.

Procedure

1. List the nodes in your cluster, along with their labels:

```
kubectl get nodes --show-labels
```

The output is similar to this (shown here in a table):

Name	Status	Roles	Age	Version	Labels
`worker 0`	Ready	<none>	1d	v1.13.0	`… ,kubernetes.io/ hostname=worker 0`
`worker 1`	Ready	<none>	1d	v1.13.0	`… ,kubernetes.io/ hostname=worker 1`
`worker 2`	Ready	<none>	1d	v1.13.0	`… ,kubernetes.io/ hostname=worker 2`

2. Choose one of your nodes, and add a label to it by using the following command:

```
kubectl label nodes <your-node-name> <aap_node_type>=<execution>
```

For example:

```
kubectl label nodes <your-node-name> disktype=ssd
```

where `<your-node-name>` is the name of your chosen node.

3. Verify that your chosen node has a `disktype=ssd` label:

```
kubectl get nodes --show-labels
```

4. The output is similar to this (shown here in a table):

Name	Status	Roles	Age	Version	Labels
`worker0`	Ready	<none>	1d	v1.13.0	`…disktype=ssd,kubernetes.io/hostname=worker0`
`worker1`	Ready	<none>	1d	v1.13.0	`…,kubernetes.io/hostname=worker1`
`worker2`	Ready	<none>	1d	v1.13.0	`…,kubernetes.io/hostname=worker2`

You can see that the `worker0` node now has a `disktype=ssd` label.

5. In the automation controller UI, specify that label in the metadata section of your customized pod specification in the container group.

```
 apiVersion: v1
kind: Pod
metadata:
  disktype: ssd
  namespace: ansible-automation-platform
spec:
  serviceAccountName: default
  automountServiceAccountToken: false
  nodeSelector:
    aap_node_type: execution
  containers:
    - image: >-
     registry.redhat.io/ansible-automation-platform-22/ee-supported-
rhel8@sha256:d134e198b179d1b21d3f067d745dd1a8e28167235c312cdc233860410ea3ec3e
      name: worker
      args:
        - ansible-runner
        - worker
        - '--private-data-dir=/runner'
      resources:
        requests:
          cpu: 250m
          memory: 100Mi
```

Extra settings

With `extra_settings`, you can pass many custom settings by using the awx-operator. The parameter `extra_settings` is appended to `/etc/tower/settings.py` and can be an alternative to the `extra_volumes` parameter.

Name	Description	Default
`extra_settings`	Extra settings	''

Example configuration of `extra_settings` parameter

```
spec:
   extra_settings:
     - setting: MAX_PAGE_SIZE
       value: "500"

     - setting: AUTH_LDAP_BIND_DN
       value: "cn=admin,dc=example,dc=com"

     - setting: SYSTEM_TASK_ABS_MEM
       value: "500"
```

Custom pod timeouts

A container group job in automation controller transitions to the `running` state just before you submit the pod to the Kubernetes API. Automation controller then expects the pod to enter the `Running` state before `AWX_CONTAINER_GROUP_POD_PENDING_TIMEOUT` seconds has elapsed. You can set `AWX_CONTAINER_GROUP_POD_PENDING_TIMEOUT` to a higher value if you want automation controller to wait for longer before canceling jobs that fail to enter the `Running` state. `AWX_CONTAINER_GROUP_POD_PENDING_TIMEOUT` is how long automation controller waits from creation of a pod until the Ansible work begins in the pod. You can also extend the time if the pod cannot be scheduled because of resource constraints. You can do this using `extra_settings` on the automation controller specification. The default value is two hours.

This is used if you are consistently launching many more jobs than Kubernetes can schedule, and jobs are spending periods longer than `AWX_CONTAINER_GROUP_POD_PENDING_TIMEOUT` in *pending*.

Jobs are not launched until control capacity is available. If many more jobs are being launched than the container group has capacity to run, consider scaling up your Kubernetes worker nodes.

Jobs scheduled on the worker nodes

Both automation controller and Kubernetes play a role in scheduling a job.

When a job is launched, its dependencies are fulfilled, meaning any project updates or inventory updates are launched by automation controller as required by the job template, project, and inventory settings.

If the job is not blocked by other business logic in automation controller and there is control capacity in the control plane to start the job, the job is submitted to the dispatcher. The default settings of the "cost" to control a job is 1 *capacity*. So, a control pod with 100 capacity is able to control up to 100 jobs at a time. Given control capacity, the job transitions from *pending* to *waiting*.

The dispatcher, which is a background process in the control plan pod, starts a worker process to run the job. This communicates with the Kubernetes API using a service account associated with the container group and uses the pod specification as defined on the Container Group in automation controller to provision the pod. The job status in automation controller is shown as *running*.

Kubernetes now schedules the pod. A pod can remain in the *pending* state for `AWX_CONTAINER_GROUP_POD_PENDING_TIMEOUT`. If the pod is denied through a `ResourceQuota`, the job starts over at *pending*. You can configure a resource quota on a namespace to limit how many resources may be consumed by pods in the namespace. For further information about ResourceQuotas, see Resource Quotas.

Minimize downtime during OpenShift Container Platform upgrade

Make the following configuration changes in automation controller to minimize downtime during the upgrade.

Before you begin

- Ansible Automation Platform 2.4 or later
- Ansible automation controller 4.4 or later
- OpenShift Container Platform:
  - Later than 4.10.42
  - Later than 4.11.16
  - Later than 4.12.0
- High availability (HA) deployment of Postgres
- Multiple worker nodes that automation controller pods can be scheduled on

Procedure

1. Enable `RECEPTOR_KUBE_SUPPORT_RECONNECT` in AutomationController specification:

```
 apiVersion: automationcontroller.ansible.com/v1beta1
kind: AutomationController
metadata:
  ...
spec:
  ...
  ee_extra_env: |
    - name: RECEPTOR_KUBE_SUPPORT_RECONNECT
      value: enabled
    ```
```

2. Enable the graceful termination feature in AutomationController specification:

```
termination_grace_period_seconds: <time to wait for job to finish>
```

3. Configure `podAntiAffinity` for web and task the pod to spread out the deployment in AutomationController specification:
```

```
task_affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - podAffinityTerm:
          labelSelector:
            matchExpressions:
            - key: app.kubernetes.io/name
              operator: In
              values:
              - awx-task
          topologyKey: topology.kubernetes.io/zone
        weight: 100
  web_affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - podAffinityTerm:
          labelSelector:
            matchExpressions:
            - key: app.kubernetes.io/name
              operator: In
              values:
              - awx-web
          topologyKey: topology.kubernetes.io/zone
        weight: 100
```

4. Configure `PodDisruptionBudget` in OpenShift Container Platform:

```
---
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: automationcontroller-job-pods
spec:
  maxUnavailable: 0
  selector:
    matchExpressions:
      - key: ansible-awx-job-id
        operator: Exists
---
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: automationcontroller-web-pods
spec:
  minAvailable: 1
  selector:
    matchExpressions:
      - key: app.kubernetes.io/name
        operator: In
        values:
          - <automationcontroller_instance_name>-web
---
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: automationcontroller-task-pods
spec:
  minAvailable: 1
  selector:
    matchExpressions:
      - key: app.kubernetes.io/name
        operator: In
        values:
          - <automationcontroller_instance_name>-task
```

How job capacity is determined and impacts job runs

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

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

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

Instance groups, not instances themselves, can be assigned to be used by jobs at various levels.

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

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

Related information
Capacity settings for instance group and container group
Job slice execution behavior

Resource determination for capacity algorithm

Capacity algorithms determine how many forks a system is capable of running simultaneously. These algorithms control how many systems Ansible can communicate with simultaneously.

Increasing the number of forks an automation controller system is running enables jobs to run faster by performing more work in parallel. However, this increases the load on the system, which can cause work to slow down.

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

Memory relative capacity

The memory relative capacity option allows you to set the maximum number of concurrent tasks (forks) that can run on a controller based on the amount of memory available on the system. This setting is useful for systems where memory is a limiting factor for running Ansible jobs.

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

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

The following is an example:

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

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

CPU relative capacity

automation controller uses the `cpu_capacity` algorithm to determine the relative CPU capacity of managed nodes. This information is used to optimize the distribution of tasks across the available nodes.

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

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

```
cpus * fork_per_cpu
```

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

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

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

Job type impact on capacity

When configuring automation controller capacity, it is important to understand how different job types impact the system capacity.

automation controller uses Ansible to run jobs. Each job can have a different impact on system resources depending on the number of forks used for the job.

The default forks value for Ansible is five. This means that, by default, each job can run tasks on up to five systems concurrently.

However, if you set up automation controller to run against fewer systems than that, then the actual concurrency value is lower.

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

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

Impact of job types in automation controller

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

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

Job types including the following, have a fixed impact:

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

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

Select the correct capacity

When an instance is created, automation controller calculates the capacity of the instance based on two algorithms: CPU-bound and memory-bound. The CPU-bound algorithm calculates the number of forks based on the number of CPU cores available to the instance.

The memory-bound algorithm calculates the number of forks based on the amount of memory available to the instance. By default, automation controller selects the minimum number of forks calculated by these two algorithms. This is to ensure that the instance does not overcommit resources. However, in some cases, you might want to adjust this behavior.

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

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

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

Procedure

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

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

Fine-tune Receptor worker backoff strategies for API reliability

Configure the Receptor worker within the Ansible Automation Platform Operator through the `RECEPTOR_KUBE_RETRY_COUNT` environment variable. This variable controls how the worker handles Kubernetes API connection failures.

NOTE:
The retry mechanism uses an exponential backoff strategy which is capped at 5 minutes to prevent excessive wait times during job execution errors.

RECEPTOR_KUBE_RETRY_COUNT details

Variable	Description	Default value	Valid range
`RECEPTOR_KUBE_RETRY_COUNT`	Sets the maximum number of retry attempts for Kubernetes API operations within the Receptor worker. Retry delays increase using exponential backoff with a Fibonacci-like sequence.	5	1-100

Scale up to an enterprise topology

Growth topologies are suited to proof-of-concept deployments, small-scale environments, or preliminary evaluations. Using a growth topology simplifies initial setup for your Ansible Automation Platform deployment, but they have inherent limitations.

Inherent limitations of growth topologies

Growth topologies include single points of failure, such as a single platform gateway , and other critical components, such as the control plane, execution plane, and web services. These components often share resources on the same node, resulting in resource contention under increasing load. As workloads grow, specific services, such as job processing or API responsiveness, can become bottlenecks due to colocation or single node capacity limits. Consequently, growth topologies generally do not offer robust, high-availability capabilities. For VM-based installation and containerized deployments of Ansible Automation Platform, you can marginally increase possible workloads by vertically scaling the virtual machines or physical hosts within the growth deployment. However, vertical scaling capabilities within a growth topology are limited.

Use cases for migrating to an enterprise topology

To scale beyond the limitations within growth topologies, you can migrate to an enterprise topology. Migrating to an enterprise topology might be relevant in the following use cases:

- Vertically scaling a growth topology becomes impractical due to cost or availability.
- The growth topology cannot satisfy high availability and disaster recovery requirements.
- You must scale Ansible Automation Platform services independently, such as API handling, job execution, and database capacity.
- Workload demands consistently overwhelm the capacity of vertically scaled growth topologies.
- You require more complex network architectures, such as segmented networks.

Recommended enterprise topology

To maximize flexibility, resilience, and scalability, migrate to the OpenShift Container Platform-based enterprise topology. This migration includes integration with an externally managed, enterprise-grade PostgreSQL database. operator-based installation offers greater flexibility to scale individual services and adapt the deployment to specific requirements. They also enhance the ability to scale the deployment up and down with reduced downtime and customize workload placement with labels, taints, tolerations, and topology constraints. operator-based installation also benefits from resilience features, such as automatic service recreation if underlying worker nodes experience failure or resource exhaustion.

Motivations for customizing enterprise topologies

Enterprise topologies provide a pattern for scalability and resilience. Organizations typically evolve the tested deployment models into custom deployments, tailoring service configurations and scaling to their specific workflows and performance needs within Red Hat Ansible Automation Platform.

An organization’s unique use of Ansible Automation Platform determines which components require scaling, moving from a generic enterprise topology to a workload-tuned deployment.

Workloads such as infrequent automation hub use or API-heavy integrations necessitate different scaling priorities. This includes components such as the API service or execution plane.

Motivations for customizing the documented enterprise deployment models include the following:

- Achieving high availability
- Enabling independent scaling of components, such as automation controller API compared to execution capacity
- Supporting workload growth or specific SLAs

This requires custom resource allocation and performance tuning based on identified needs, rather than adherence to a general pattern.

Before customizing and scaling, you must identify specific bottlenecks within your Ansible Automation Platform environment, such as:

- API response
- Job processing
- Database performance
- Event-Driven Ansible event handling.

Use platform monitoring tools and analytics to identify bottlenecks. After bottlenecks are identified, you can approach scaling each component vertically or horizontally.

API request flow and latency sources for performance tuning

The Ansible Automation Platform API uses distributed services. Performance depends on the request path, with latency possible at each layer. The following table details each layer in the API request flow:

API Service architecture and performance considerations

Layer	Description	Performance Consideration
Client Request	The request from the client.	The request from the client might have timeout parameters set. In the VM-based installation and containerized installation program, a variable `client_request_timeout` is used to inform downstream component timeouts. This value must match the timeout of the external load balancer. The size of the request body and headers can also impact performance.

Layer	Description	Performance Consideration
Ingress Point	The first point of entry into Ansible Automation Platform, typically an OpenShift Container Platform Route or a customer-provided Load Balancer, directing traffic to an available platform gateway pod/instance.	Performance is dependent on the configuration, capacity, and health of the load balancer or OpenShift Container Platform Route. Any timeout for the external load balancer must be greater than or equal to the `client_request_timeout` setting passed to the installation program. This layer is responsible for distributing traffic if there are multiple platform gateway nodes/pods.
Envoy Proxy	Located within the platform gateway pod/instance, this proxy handles authorization, internal routing, and applies filters to the request. Authorization by the gRPC service is performed before the Envoy forwards the request to the destination service.	Introduces minimal latency, typically on the order of 10 milliseconds. Can handle hundreds of concurrent requests.
Platform gateway gRPC Authentication Service	A local gRPC service within the platform gateway container responsible for authenticating each request. This service can interact with external authentication services (LDAP/SAML) and the database for validation. Authentication with the gRPC service can be disabled for individual URL routes, notably requests to the platform gateway service itself are not authenticated by this gRPC service (for example, under `/api/gateway/v1/`). These requests are authenticated by the platform gateway API service itself.	Potential source of latency. The service is multi-processed and multi-threaded, with capacity determined by `GRPC_SERVER_PROCESSES` and `GRPC_SERVER_MAX_THREADS_PER_PROCESS`. If all workers are busy, then requests queue, which adds to latency. In containerized or VM-based installation, its timeout is informed by the `client_request_timeout`. Slow database connections for session validation also negatively impact performance.
External Authentication Service (LDAP/SAML)	An optional external service invoked by the platform gateway gRPC Authentication Service for validating user credentials.	Potential source of latency. When external authentication services (e.g., LDAP or SAML) are configured, they are invoked during the gRPC authentication stage. Slow response times from these external systems can significantly increase the total latency for each request processed. It is the user's

Layer	Description	Performance Consideration
		responsibility to provide a low-latency, reliable external authentication service.
API Service Nginx Proxy	After authentication, Envoy forwards the request to the component API node or Service in OpenShift Container Platform. Nginx receives the request. Each distinct API service component has its own Nginx proxy that determines if the request is for a WSGI application, an ASGI-based WebSocket service, or static content.	Introduces minimal latency, typically on the order of 10 milliseconds. Can handle hundreds of concurrent requests.
WSGI Server (`uWSGI` / `Gunicorn`)	Handles standard API requests forwarded by Nginx. These servers process requests, validate JWT tokens, run API operations, and frequently interact with the database.	Primary source of latency. API requests are handled by each component's web application served by a WSGI server (`uWSGI` for automation controller and platform gateway and `Gunicorn` for automation hub and Event-Driven Ansible), and their timeout is also informed by the `client_request_timeout` in VM-based installation and container-based installation. In OpenShift Container Platform, the timeout on the platform gateway Route is propagated to inform this same setting. The servers are configured with a maximum number of concurrent workers. If all workers are busy, the request is queued. After a worker picks up a request, it validates the authentication and executes the API operation, which typically involves further database communication.
Databases	Almost every request requires interacting with the database to do activities such as validating sessions, storing data, and executing API operations.	Critical performance factor. Almost every request requires interacting with the database. The responsiveness of database connections remains a critical factor in API performance, impacting both the gRPC authentication service and the WSGI server processing. Responsiveness can be impacted by network latency between the database and components, as well as performance of the database itself.

Layer	Description	Performance Consideration
Client Response	The final response returned to the client after the request has been processed and traversed back through the system components (Nginx proxy, Envoy, and the initial load balancer/Route).	The final response returned to the client after the request has been processed and traversed back through the system components (Nginx proxy, Envoy, and the initial load balancer/Route).

Sources of latency and scaling strategies

The primary sources of latency across all layers are:

- Queueing delays while awaiting an available worker from either the gRPC authentication service or the WSGI server
- The authentication phase, particularly if external authentication systems exhibit slow response times
- The actual processing time and associated database interactions within the Python WSGI application

Scaling strategies include the following:

- Using more performant authentication methods, such as Session or Token
- Horizontally scaling platform gateway and API service pods to increase worker availability and minimize queue times

The following sections describe how to identify which of the Ansible Automation Platform services provide which APIs and provide considerations for scaling them. For more information on the performance of different authentication methods, see Considerations for scaling the platform gateway proxy and authentication service.

Key performance indicators to guide scaling plans

Scaling adds resources to handle increased load. This is primarily achieved through horizontal scaling (adding more pods or instances) or vertical scaling (adding CPU or memory resources to pods or instances). Proper scaling ensures high availability and maintains performance under load.

Scale your services when key performance indicators suggest a component is reaching capacity. These indicators show the component cannot efficiently handle the current request load and include the following:

- High API latency
- High CPU utilization
- Errors that occur during periods of high traffic

High API latency

Sustained high latency on API requests is a key performance indicator. All requests are made through the platform platform gateway, which acts as a proxy and forwards requests to the services in question. The request is sent to the destination service depending on which route is in the URL of the API request:

- platform gateway: `/api/gateway`
- automation controller: `/api/controller`
- Event-Driven Ansible: `/api/eda`
- Event-Driven Ansible Event Streams: `/eda-event-streams/api/eda/v1/external_event_stream/`
- automation hub: `/api/galaxy`

Monitoring latency on the different routes through the Envoy proxy logs helps you identify which service requires scaling. These routes are present in the proxy container of platform gateway pods in OpenShift Container Platform. For VM-based installation and container-based installation installations, check the proxy logs of the platform gateway nodes. Exceeding target API thresholds (for example, 99th percentile >1500ms) indicates a need to trigger alerts. You might also need to scale web services.

High CPU utilization

When a service's API pod shows consistently high CPU usage, it might be unable to process incoming requests promptly. This can lead to a backlog of requests. The following indicators suggest high CPU utilization:

- High total request time from the Envoy proxy logs with the processing time from the service's WSGI logs
- High total Envoy latency
- Requests are waiting in a queue before being processed

Error codes

Error codes indicate the service must be scaled. Find these codes in the platform gateway's proxy container (on OpenShift Container Platform) or in the proxy logs (for VM-based and container-based installations). They are often precipitated by the services being overloaded and unable to service requests in a timely manner, and are often preceded by periods of higher latency.

- Upstream Authentication Failures: `502 UAEX` (Upstream Authentication Extension) indicates issues during the authentication phase of a request. This suggests the authentication service is overloaded, timing out, or returning broken responses.
- Upstream Service Unhealthy: A `503 UH` (Upstream Service Unhealthy) response means Envoy has marked one or more service pods as unhealthy. Traffic is not being sent to that unhealthy pod.
- Upstream Connection Failure: `503 UF` (Upstream Connection Failure) response indicates a connection failure. Envoy attempted to contact an upstream pod, but the connection failed. For more information about Envoy Response Flags (the letter codes that follow the `HTTP` response code), see *Access logging* in the Related Links section.

Related information
Access logging

Scale the platform gateway proxy and authentication service

Scale the platform gateway proxy and authentication services if request volume exceeds capacity. Horizontal scaling is preferred. Vertical scaling does not automatically adjust worker pools for gRPC, Envoy, Nginx, and WSGI services.

NOTE:
Additional platform gateway service pods or instances can increase the necessary database connections for WSGI web service workers and gRPC authentication service workers.

If you observe a bottleneck in CPU utilization, then scaling the gRPC authentication service can improve throughput. However, if external authentication providers are the source of high latency, then horizontal scaling of the gRPC service has minimal benefits. You can determine the gRPC authentication latency for requests by observing the difference between the upstream service time and the total request latency.

The most performant authentication methods are:

- Token authentication

- Session authentication
- Basic authentication must not be used for high-frequency API automation because CPU-intensive password hashing introduces significant request latency. If Basic authentication is used in combination with LDAP authentication, reaching out to the LDAP service can introduce significant latency, especially if the LDAP service has limited availability. For this reason, we recommend creating OAuth Tokens to perform automation against the API.

Horizontally scaling the platform gateway service pods also increases the number of health checks to each component's API, because each Envoy tracks this separately. You can observe these in logs with the user agent `Envoy/HC`. Since health checks flow through the same services and queues as user-initiated requests, if overall request times slow due to overload, health checks can also timeout. When this occurs, Envoy stops forwarding requests to these service nodes until they pass their health check again.

Special considerations for scaling the platform gateway proxy and authentication service on OpenShift Container Platform

It is particularly important that the service is horizontally scaled sufficiently in OpenShift Container Platform, because if more than 100 requests are backlogged, then these requests are then dropped by uWSGI. This results in clients receiving a timeout for dropped requests. The following log text provides the corresponding error for this event:

```
*** uWSGI listen queue of socket ":8000" (fd: 3) full !!! (101/100) ***
```

This error occurs due to a limitation of uWSGI tying its backlog length to the kernel parameter `somaxconn`. It is possible to raise this kernel parameter in OpenShift Container Platform, but doing so requires allowing "unsafe sysctls".

Scale the automation controller API service

The automation controller API service handles HTTP requests to the application, including information about user roles in automation controller, project creation, inventory creation or updates, job launches, and job result checks.

Key performance indicators

Key performance indicators for the automation controller API service include the following:

- High API latency for requests under `/api/controller`
- High CPU utilization on the API pods or nodes
- Platform gateway returning `503` errors because the service is too busy to respond to health checks

The automation controller API service is located in web pods on operator-based installation and in control or hybrid nodes on VM-based installation or container-based installation.

Scaling strategies by deployment type

Consider the following strategies to scale the automation controller API service:

- OpenShift Container Platform: Adjust the `web_replicas` attribute on the `AutomationController` CR. Scaling the `replicas` attribute scales task and web replicas.
- VM-based installation and container-based installation: Scale control or hybrid nodes, increasing the ability to control additional automation jobs.

Database connection and architecture considerations

On OpenShift Container Platform, each web replica consumes database connections for WSGI web service workers and various background services facilitating task communication and WebSockets. The number of database connections used by the WSGI web server on VM-based installation and container-based installation scales with the machine's CPU count.

Additionally, control and hybrid nodes manage the Dispatcher (tasking system) and the Callback Receiver (job event processing worker pool). These worker pools scale with CPU availability and necessitate database connections.

Provisioning additional control nodes demands more database connections than scaling out the web deployment on OpenShift Container Platform alone. This demand occurs because containerized and RPM control node scaling also expands the tasking system, which operates as a distinct deployment on OpenShift Container Platform. This separation of services on OpenShift Container Platform deployments is an important distinction that allows administrators to more finely tune the deployment and conserve limited resources, such as database connections.

Special considerations for scaling on OpenShift Container Platform

It is particularly important that the service is horizontally scaled sufficiently in OpenShift Container Platform, because if more than 100 requests are backlogged, then these requests are then dropped by uWSGI. This results in clients receiving a timeout for dropped requests. The following log text provides the corresponding error for this event:

```
*** uWSGI listen queue of socket ":8000" (fd: 3) full !!! (101/100) ***
```

This error occurs due to a limitation of uWSGI tying its backlog length to the kernel parameter `somaxconn`. It is possible to raise this kernel parameter in OpenShift Container Platform, but doing so requires allowing "unsafe sysctls".

Scale the Event-Driven Ansible API service

Learn how to scale the Event-Driven Ansible API service.

Scaling Event-Driven Ansible involves considerations for each of its service types:

- API and WebSocket service
- EventStream service

API requests routed to `/api/eda` and `/api/eda-event-stream` are handled by two separate `Gunicorn` deployments. In OpenShift Container Platform, these services must be scaled independently. For VM-based installation and container-based installation, you can scale these services together by increasing the number of hybrid nodes.

Event-Driven Ansible API and WebSocket service

The Event-Driven Ansible API service handles `HTTP` requests to the application, including information about user roles in Event-Driven Ansible, creating projects, creating activations, and checking results. Key performance indicators for the API service include the following:

- High API latency for requests under `/api/eda`
- High CPU utilization on the API pods/nodes

- Platform gateway returning `503` errors because the service is too busy to respond to health checks

A WebSocket service is deployed alongside the API service to manage the output of the rulebook activations. Each activation maintains a persistent WebSocket connection to this service to communicate status and receive instructions. A large number of activations or a large amount of output from activations can overwhelm the WebSocket server, leading to failures.

Consider the following strategies to scale the Event-Driven Ansible API and WebSocket service:

- OpenShift Container Platform: The WebSocket server (Daphne) runs within the `eda-api` pod. Horizontally scale the `eda-api` deployment to scale this service.

> **NOTE:**
> Ensure that you scale the `eda-api` deployment in proportion to the number of activations being run.

- VM-based installation or container-based installation: Horizontally scale the WebSocket service alongside the API service by adding more hybrid nodes. This increases capacity for all Event-Driven Ansible components simultaneously.

To identify whether your deployment experienced a bottleneck in the WebSocket service, check the activation logs for the following errors:

```
ansible_rulebook.websocket - WARNING - websocket aborted by <class
'websockets.exceptions.InvalidMessage'>: did not receive a valid HTTP response

ansible_rulebook.cli - ERROR - Terminating: did not receive a valid HTTP response
```

Event-Driven Ansible EventStream service

The Event Stream API, which handles POST requests to `/api/eda-event-stream`, is a separate `Gunicorn` service designed to import events from external sources. If this service's performance degrades, with high latency, low throughput, or availability issues, you must scale it. The platform gateway returns `503` errors for this service when a pod is too busy to respond to health checks in time.

Consider the following strategies to scale the Event-Driven Ansible EventStream service:

- OpenShift Container Platform: Horizontally scale up the dedicated `eda-event-stream` worker deployment, because it is managed separately from the main `eda-api deployment`.
- VM-based installation or container-based installation: Horizontally scale up this service by adding more hybrid nodes, which increases capacity for all Event-Driven Ansible components simultaneously.

Scale the automation hub API service

Learn about scaling the automation hub API service.

Scaling automation hub involves considerations for each of its service types:

- API service: manages `HTTP` requests through the API
- Pulp workers service: manages syncs and content uploads
- Content service: manages content delivery after content has been processed and stored

Separate `Gunicorn` deployments back these services and handle different types of requests. In OpenShift Container Platform, these services must be scaled independently. In VM-based installation and container-based installation, a standard automation hub node hosts all services, and scaling is achieved by adding more nodes.

Automation hub API service

The automation hub API service handles metadata-driven requests for the application, including UI interactions, searches, and remote repository configuration. Key performance indicators for the automation hub API service include:

- High API latency for requests under `/api/galaxy`
- High CPU utilization on the API pods or nodes
- Platform gateway returning `503` errors because the service is too busy to respond to health checks

Consider the following strategies to scale the automaton automation hub API service:

- OpenShift Container Platform: Horizontally scale the API pods by increasing the `hub.api.replicas` attribute on the `AnsibleAutomationPlatform` Custom Resource (CR).
- VM-based installation or container-based installation: Horizontally scale the API service by adding more automation hub nodes, which simultaneously scales all other automation hub services.

Automation hub Pulp worker and content services

The Pulp worker and content services manage all operations related to content syncs, uploads and downloads. Key performance indicators for the Pulp worker and content services include:

- High Content sync rates: Frequent or large synchronization operations from external repositories demanding significant worker processing.
- High Content upload or download rates: Frequent pushing or pulling of automation execution environments by automation controller, Event-Driven Ansible, or large Collection uploads or downloads by automation clients.
- Disk I/O bottlenecks: Performance issues related to read/write operations on the underlying content storage (`/var/lib/pulp`), often shown as high disk I/O wait times.
- Pulp worker saturation: High CPU utilization or queuing within pulp processes, indicating an inability to keep up with content processing and serving.

To scale your Pulp worker and content services, consider the following scaling strategies:

- In OpenShift Container Platform: Scale the deployment of these services by increasing the `hub.content.replicas` and `hub.worker.replicas` attributes on the `AnsibleAutomationPlatform` Custom Resource.
- For VM-based installation or container-based installation: Horizontally scale all services by adding more automation hub nodes.

Tune automation controller to improve performance

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

Configure WebSocket load balancing

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 WebsSocket 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.

Configure automatic discovery of other automation controller nodes

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

Procedure

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
```

Capacity plan for node types and workload characteristics

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

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 decide 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)

Types of nodes in automation controller

Learn about the 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. As with 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 keeping 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

Hop nodes offer a communication bridge between the control plane and execution nodes, especially where execution nodes are located in a different network segment or behind firewalls.

Because hop nodes use very low memory and CPU, vertically scaling these nodes does not impact capacity. Check 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 offer 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

When planning your deployment architecture, consider the ratio of control capacity to execution capacity. Control capacity refers to the resources allocated to the control plane, while execution capacity refers to the resources allocated to run jobs.

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.

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.

Example workload requirements

Learn how to calculate workload requirements for automation controller based on a hypothetical workload.

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

Execution capacity

The minimum execution capacity required is 60 units to support 10 concurrent jobs.

Derive the total capacity by summing the resource consumed by parallel execution forks and the base resource consumed by each job:

(10 x jobs * 5 x forks) + (10 x jobs * 1 x base task impact of a job) = (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 tasks x 300 managed hosts per hour = 300,000 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

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.

Using this default, the maximum number of jobs a control node can dispatch to execution nodes makes 5 execution nodes of equal CPU and RAM use 100% of their capacity, arriving to the 1:5 ratio of control to execution capacity mentioned before.

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.

Troubleshoot common performance issues

This section provides guidance on troubleshooting common performance issues with automation controller.

Common performance issues and their resolution

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.

- 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.

Understand 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

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. Project update jobs in the API manage synchronization. These jobs are also known as source control updates in the UI.

These project update jobs run only 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. This content comes from its defined source, such as a Git repository.

Updating projects is not performance-sensitive, provided that they store only playbooks and Ansible-related text files. However, issues might arise when projects become excessively large.

Do not store large volumes of binary data within a project. If jobs require access to additional data, they should retrieve it from object storage or file storage. This retrieval must be done within the playbook's scope.

Jobs and automation workloads

Jobs are the primary workload for automation controller and 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
- 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 usage. 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

To enable complex automation and orchestration, use the following job types to extend standard jobs:

- 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 run 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 to retain can degrade database performance. It is generally recommended to retain as few days of data as possible. Use external logging features for long-term audit data storage. Storing more data in the database can make queries that scan large tables more costly.

Tune 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.

Activation and output management uses the following:

- Event-Driven Ansible hybrid nodes
- Platform gateway for event stream handling
- The WebSocket server in each API node or pod
- The database for audit event storage

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. However, the fire count in the API and UI is updated at a periodic interval (default 300 seconds) rather than immediately.

Minimize the impact of collection syncing

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 workers and the database synchronize the repositories. The automation controller can download these collections during project updates, or use them to build automation execution environments. Collections are also available for any other client by using the `ansible-galaxy` CLI to download and use.

The performance of collection synchronization is impacted by the following:

- The number of collections listed in the `requirements.yml`

- The number of versions synced
- The number of versions retained

Synchronization uses memory in direct proportion to the number of collections and versions synchronized. Using a targeted `requirements.yml` with specific versions can limit this impact.

Hosting collections uses storage space. Manage the storage space that collections use by specifying the retained number of versions on the repository.

Pull hosted container images from private automation hub

Private automation hub hosts container images for automation execution and decision environments. Event-Driven Ansible and automation controller pull these images to run activations or jobs. The pull frequency for these containers is determined by the following:

- The frequency of job starts
- The pull policy configured for the automation execution environments and decision environments

The performance of pushing and pulling container images from automation hub depends on the disk performance of the underlying storage. This is 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 content workers. This is because they serve entire layers in a single operation.

The frequency of container image pulls is determined by the following factors:

- The pull policy on jobs and activations
- The frequency of job or activation starts
- The node or Container Group's existing image status

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.

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
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

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.

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
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

Manage live event streams to the UI

By default, automation controller streams live events to the user interface (UI) for jobs that are running.

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.

Result

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.

Disable live streaming events

You can disable live streaming events in automation controller to reduce system load or for troubleshooting purposes.

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. In the navigation panel, select **Settings > Automation Execution > System**.
        i. Click **Edit**.
        ii. Set the **Enable Activity Stream** option to **Off**.

Settings to modify rate and size of events

If your system generates a large number of events, the live streaming of events to the user interface (UI) can cause performance issues. You can disable live streaming of events or reduce the number of events that are displayed in the UI by modifying the following settings.

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.

Capacity settings for each node type

Configure capacity settings for control and execution nodes to manage resource allocation effectively.

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 required 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.

Capacity settings for instance group and container group

When using container groups or instance groups to run automation jobs, you can configure capacity settings to limit the number of concurrent jobs and forks that can be used.

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:

```
((number of worker nodes in kubernetes cluster) * (CPU available on each
worker)) / (CPU request on pod_spec) = 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:

```
((number of worker nodes in kubernetes cluster) * (memory available on each
worker)) / (memory request on pod_spec) = 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.

Adjust settings for scheduling jobs

The controller uses a task manager to schedule jobs onto instances.

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.

VM installation settings for 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 several 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 installation program reaches the host, which in this case is an internal IP address. This is not used outside of the installation program.

- `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 can cross many networks, we are using a hostname that maps to an IP address resolvable for the receptor peers.

Tune the web server

The 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.

Related information
Scaling Automation Controller for API Driven Workloads
Bulk API in Automation Controller
Configuring access to external applications with token-based authentication

Tune the PostgreSQL database for optimal performance

Improve the performance of automation controller, by configuring 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 `$PDATA` 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. f 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.

Related information
Automatic Vacuuming

Encrypt plain text passwords in automation controller configuration files

Plain text passwords in automation controller configuration files pose a potential security risk.

Configuration files are kept in the /etc/tower/conf.d/ folder. Files used to reach the database, for example, save passwords without encryption. This means that anyone who can read this folder can see the passwords clearly.

While permissions protect these folders, some security reports say this protection is good inadequate. The fix is to encrypt each of these passwords separately.

Encrypt the PostgreSQL password

Learn how to encrypt the PostgreSQL password used by automation controller for database connections.

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 `$encrytpted..` with your own hash value:

```
 decrypt_value(get_encryption_key('value'),'$encrypted$AESCBC$Z0FBQUFBQmNONU9Bb
GQ1VjJyNDJRVTRKaFRIR09Ib2U5TGdaYVRfcXFXRjlmdmpZNjdoZVpEZ21QRWViMmNDOGJaM0dPeHN
2b194NUxvQ1M5X3dSc1gxQ29TdDBKRkljWHc9PQ==''),
```

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$Z0FBQUFBQmNONU9Bb
GQ1VjJyNDJRVTRKaFRIR09Ib2U5TGdaYVRfcXFXRjlmdmpZNjdoZVpEZ21QRWViMmNDOGJaM0dPeHN
2b194NUxvQ1M5X3dSc1gxQ29TdDBKRkljWHc9PQ==''), 'HOST': '127.0.0.1', 'PORT':
5432, } }+
```

Tune performance for Event-Driven Ansible

Event-Driven Ansible controller provides the interface in which Event-Driven Ansible automation performs. Tune your Event-Driven Ansible controller to optimize performance and scalability through:

- Characterizing your workload
- Performance troubleshooting
- System level monitoring

Characterize your workload

Characterize your workload by quantifying event ingestion rate and concurrent rulebook activations to effectively optimize performance and capacity.

Consider the following factors to characterize your Event-Driven Ansible controller workload:

1. Number of simultaneous rulebook activations
2. Number of events received by Event-Driven Ansible controller

Modify the default memory limit for each rulebook activation

Memory usage is based on the number of events that Event-Driven Ansible controller has to process. By default, each rulebook activation container has a 200 MB memory limit.

For example, with 4 CPU and 16 GB of RAM, one rulebook activation container with an assigned 200 MB memory limit cannot handle more than 150,000 events per minute. If the number of parallel running rulebook activations is higher, then the maximum number of events each rulebook activation can process is reduced.

If there are too many incoming events at a very high rate, the container can run out of memory trying to process the events, which will kill the container, and your rulebook activation will fail with a status code of 137.

To mitigate this status, you can modify the default memory limit for each rulebook activation *during* or *after* installation.

Procedure

1. Perform the following steps to modify your default memory limit for your rulebook activations *during* installation:
    a. Navigate to the setup inventory file.
    b. Add `automationedacontroller_podman_mem_limit` in the [all:vars] section. For example, `automationedacontroller_podman_mem_limit='400m'`.
    c. Run the setup.
2. Perform the following steps to modify your default memory limit for your rulebook activations *after* installation:
    a. Navigate to the environment file at `/etc/ansible-automation-platform/eda/settings.yaml`.
    b. Modify the default container memory limit. For example, `PODMAN_MEM_LIMIT = '300m'`.

c. Restart the Event-Driven Ansible controller services using `automation-eda-controller-service restart`.

System level monitoring for Event-Driven Ansible controller

After characterizing your workload to determine how many rulebook activations you are running in parallel and how many events you are receiving at any given point, conduct system-level monitoring to ensure the host environment can sustain the resource demands of the event-driven workload.

Using system level monitoring to review information about Event-Driven Ansible's performance over time helps when diagnosing problems or when considering capacity for future growth.

System level monitoring includes the following information:

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

Higher CPU, RAM, or Disk utilization can affect the overall performance of Event-Driven Ansible controller.

For example, a high utilization of any of these system level resources indicates that either the Event-Driven Ansible controller is running too many rulebook activations, or some of the individual rulebook activations are using a high volume of resources. In this case, you must increase your system level resources to support your workload.

Get insights on automation across your environment with Automation Analytics

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

Only users installing a trial of automation controller or a fresh installation of automation controller are opted-in for this data collection.

Related information
Red Hat privacy policy

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 Lightspeed, 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.

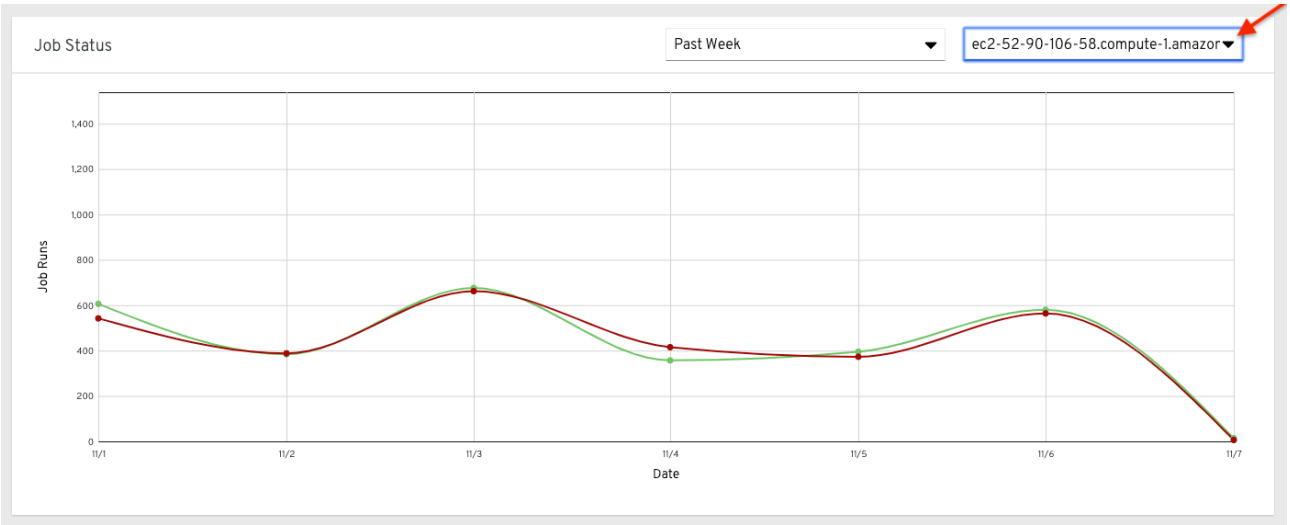
This setting can also be enabled through the API by specifying `INSIGHTS_TRACKING_STATE = true` in either of these endpoints:

- `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 and jobs that failed.

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 and jobs that failed. 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.

Configure 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.

Before you begin

- 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**.

Result

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.

Use automation calculator to determine automation savings

The automation calculator provides graphs, metrics and calculations that help you determine the total savings on your investment in automated processes.

Calculate your automation savings

The **Automation Calculator** produces its default total savings figure based on estimates for each variable.

You can tune this calculation by providing more specific organizational cost information and adjusting the time values for each of the top templates.

NOTE:
Automation savings calculations are not saved in Automation Analytics.

Procedure

1. Under **Calculate your automation** enter cost information for:
    a. **Manual process cost**
    b. **Automated process cost**
2. Under **Top templates**:
    a. Adjust time values for top templates to provide time to manually perform each task that the template automates.

Result

Total savings updates based on the information you enter in each field.

Top templates

Top templates lists the 25 most frequently run templates across all hosts in your environment.

Templates are listed in descending order starting with the highest run count. You can enter the time it takes to perform tasks manually that are automated by templates in the field adjacent to the run totals to produce a more accurate total savings. The default value is set to **60** minutes.

Curate top templates

You can use the toggle switch for each template to show or hide it in the bar graph to compare performance and savings based on specific templates.

- Click the toggle switch for each template to display or hide it.

Result

The bar graph on the **Automation Calculator** will update to display those top templates selected and **Total savings** will calculate based on those templates.

View template details

View detailed information for each template in **Top templates** to learn more about the template's context in the calculation of automation savings.

Procedure

1. Click the **Info** icon for a job template to view template details.

   Top template information includes the following:

   - **Total elapsed sum** - Total run time of the template.
   - **Success elapsed sum** - Total run time for successful template runs.
   - **Failed elapsed sum** - Total run time for failed template runs.
   - **Automation percentage** - The template accounts for this percentage of automation in your organization.
   - **Associated organizations** - The template runs against these organizations.

- **Associated clusters** - Automation controller clusters the template runs on.

Plan, track, and analyze returns with automation savings planner

An automation savings plan gives you the ability to plan, track, and analyze the potential efficiency and cost savings of your automation initiatives. Use automation analytics to create an automation savings plan by defining a list of tasks needed to complete an automation job.

You can then link your automation savings plans to an Ansible job template to accurately measure the time and cost savings upon completion of an automation job.

To create an automation savings plan, you can use the automation savings planner to prioritize the various automation jobs throughout your organization and understand the potential time and cost savings for your automation initiatives.

Create a new automation savings plan

Create an automation savings plan by defining the tasks needed to complete an automation job by using the automation savings planner.

Procedure

1. From the navigation panel, select **Automation Analytics > Savings Planner**.
2. Click **Add Plan**.
3. Provide some information about your automation job:
    a. Enter descriptive information, such as a name, description, and type of automation.
    b. Enter technical information, such as the number of hosts, the duration to manually complete this job, and how often you complete this job.
    c. Click **Next**.
4. In the tasks section, list the tasks needed to complete this plan:
    a. Enter each task in the field, then click **Add**.
    b. Rearrange tasks by dragging the item up/down the tasks list.
    c. Click **Next**.
5. Select a template to link to this plan, then click **Save**.

> **NOTE:**
> The task list is for your planning purposes only, and does not currently factor into your automation savings calculation.

Result

Your new savings plan is now created and displayed on the automation savings planner list view.

Edit an existing savings plan

Edit any information about an existing savings plan by clicking on it from the savings planner list view.

Procedure

1. From the navigation panel, select **Automation Analytics > Savings Planner**.
2. On the automation savings plan, click Click the **More Actions** icon ⋮, then click **Edit**.
3. Make any changes to the automation plan, then click **Save**.

Link a savings plan to a job template

You can associate a job template to a savings plan to allow automation analytics to provide a more accurate time and cost savings estimate for completing this savings plan.

Procedure

1. From the navigation panel, select **Automation Analytics > Savings Planner**.
2. Click the **More Actions** icon ⋮ and select **Link Template**.
3. Click **Save**.

Review savings calculations for your automation plans

The automation savings planner offers a calculation of how much time and money you can save by automating a job. Automation analytics takes data from the plan details and the associated job template to provide you with an accurate projection of your cost savings when you complete this savings plan.

To do so, navigate to your savings planner page, click the name of an existing plan, then navigate to the **Statistics** tab.

The statistics chart displays a projection of your monetary and time savings based on the information you provided when creating a savings plan. Primarily, the statistics chart subtracts the automated cost from the manual cost of executing the plan to provide the total resources saved upon automation. The chart then displays this data by year to show you the cumulative benefits for automating the plan over time.

Click between **Money** and **Time** to view the different types of savings for automating the plan.

Get a comprehensive visual overview of automation with reports

The reports feature on the Red Hat Ansible Automation Platform provides users with a visual overview of their automation efforts across different teams by using Ansible.

Each report helps users to monitor the status of their automation environment, be it the frequency of playbook runs or the status of hosts affected by various job templates.

For example, you can use your reports to:

- View the number of hosts affected by a job template
- View the number changes made to hosts by a job template
- View the frequency of a job template run, and the rate of job templates that succeed or fail to run

Review your reports

To view reports about your Ansible automation environment, proceed with the following steps:

Procedure

1. Log in to console.redhat.com and navigate to the Ansible Automation Platform.
2. Click **Reports** on the side navigation panel.
3. Select a report from the results to view it.

   Each report presents data to monitor your automation environment. Use the filter toolbar on each report to adjust your graph view.

> **NOTE:**
> New reports are regularly added to the system. If you have ideas for new reports that would be helpful for your team, contact your account representative or log a feature enhancement for automation analytics.

View data about automation jobs across your organization

The **Job Explorer** provides a detailed view of jobs run on automation controller clusters across your organizations. You can access the **Job Explorer** by selecting **Automation Analytics > Job Explorer** from the navigation panel or using the drill-down view available across each of the application's charts.

Using the **Job Explorer** you can:

- Filter the types of jobs running in a cluster or organization;
- Directly link out to templates on automation controller for further assessment;
- Identify and review job failures;
- View more details for top templates running on a cluster;
- Filter out nested workflows and jobs.

You can review the features and details of the **Job Explorer** in the following sections.

Create a filtered and sorted view of jobs

You can view a list of jobs, filtered by attributes you choose, using the **Job Explorer**.

Filter options include:

- Status
- Job

- Cluster
- Organization
- Inventory
- Template

You can sort results by any of the parameters from each column by using the directional arrows.

Procedure

1. From the navigation panel, select **Automation Analytics > Job Explorer**.
2. In the filter toolbar, select **Job** from the **Filter by** list.
3. In that same toolbar, select a time range. Job Explorer will now display jobs within that time range.
4. To further refine results, return to the filter toolbar and select a different attribute to filter results by, including job status, cluster, or organization.

   The **Job Explorer** view updates and presents a list of jobs based on the attributes you selected.

View more information about an individual job

You can view additional information about each individual job.

Procedure

1. Navigate to the job **Id/Name** column.
2. Click the arrow icon.

Review job details on automation controller

You can review job details on automation controller.

- Click the job in the **Id/Name** column to view the job itself on the automation controller job details page.

Related information
Use jobs to run playbooks against an inventory of hosts

Change your Automation Analytics settings

You can drill down into cluster data to review more detailed information about successful or failed jobs.

The detailed view, presented on the **Job Explorer** page, provides information on the cluster, organization, template, and job type. Filters you select on the **Clusters** view carry over to the **Job Explorer** page.

Details on those job templates will appear in the **Job Explorer** view, modified by any filters you select in the **Clusters** view.

For example, you can drill down to review details for failed jobs in a cluster.

Example: Review failed jobs

You can view detailed information about failed jobs across your organization by drilling down on a graph in the **Clusters** view and using the **Job Explorer** to refine results. Clicking a segment on the graph opens that information in the **Job Explorer**, preserving filters applied in the **Clusters** view.

Procedure

1. From the navigation panel, select **Automation Analytics > Clusters**.
2. Use the filter lists in the toolbar to apply filters for clusters and time range of your choosing.
3. Click a segment on the graph.

   You are redirected to the **Job Explorer** view, which displays a list of jobs corresponding to that day on the graph.
4. To refine the list to show only failed jobs:
   a. Select **Status** from the **Filter by** list.
   b. Select the **Failed** filter.

      The **Job Explorer** view updates to show only failed jobs run within the selected time range.

Next steps

- Add additional context to the view by applying more filters and selecting attributes to sort results.
- Review more information for failed jobs on the automation controller job details page.

View top templates job details for a specific cluster

You can view job instances for top templates in a cluster to learn more about individual job runs associated with that template or to apply filters to further drill down into the data.

Procedure

1. From the navigation panel, select **Automation Analytics > Clusters**.
2. Click a template name in **Top Templates**.
3. Click **View all jobs** in the modal that is displayed.

   The **Job Explorer** page displays all jobs on the chosen cluster associated with that template.

   The view presented will preserve the contextual information of the template based on the parameters selected in the **Clusters** view.

Ignore nested workflows and jobs

Ignoring nested workflows and jobs filters out duplicate workflow and job template entries and excludes those items from overall totals.

Procedure

1. Select the settings icon on the **Job Explorer** view.
2. Use the toggle switch to **Ignore nested workflows and jobs**.

> **NOTE:**
> **About nested workflows** Nested workflows enable you to create workflow job templates that call other workflow job templates. Nested workflows promotes reuse, as modular components, of workflows that include existing business logic and organizational requirements in automating complex processes and operations. To learn more about nested workflows, see Workflows in automation controller in the Using automation execution.

Details about data collected for Automation Analytics

Learn about the specific data that automation controller collects and sends to Red Hat when you enable Automation Analytics.

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.

Automation Analytics Data Dictionary

Automation Analytics Data is sent to the Red Hat Hybrid Cloud Console (HCC) to provide detailed analytics on your automation.

The data dictionary outlines the information collected by Automation Analytics from the Red Hat Ansible Automation Platform automation controller, also known as Automation Execution.

Related information
Ansible Automation Platform - Data Dictionary

View automation job metrics with automation dashboard

Track automation job performance and demonstrate automation value with automation dashboard. Analyze job execution metrics and calculate cost savings to communicate the impact of your automation initiatives to stakeholders.

Viewing automation job metrics helps you to:

- **Monitor automation execution**: Track successful and failed jobs, unique hosts automated, and total automation hours to understand platform usage patterns.
- **Analyze automation adoption**: View top projects and users to identify high-value automation workflows and understand which teams drive automation adoption.
- **Demonstrate automation value**: Demonstrate return on investment to stakeholders by calculating cost savings from automation.

Filter and save automation data for reporting

Automation dashboard provides filtering options to analyze your Ansible Automation Platform automation runs. You can select one or more filtering options to customize your report, select a time period and a currency, and save your report to your automation dashboard.

Filters

Use the following filtering options to customize your report:

- **Template:** select one or more Job Templates
- **Organization:** select one or more Organizations
- **Project:** select one or more Projects
- **Label:** select one or more automation projects by label.

NOTE:
You must preconfigure and assign labels to Ansible Automation Platform for display in automation dashboard.

Time period and currency

After selecting your filters, select a time period for analysis, and select a currency to show automation savings.

- Use a shorter time period to analyze specific automation use cases.
- Use a longer time period to evaluate overall platform usage and automation growth.
- Changing the currency does not convert the values. You must manually update the **Monthly AAP cost** and **Hourly rate** to reflect the selected currency.

Save a report

Use **Save as Report** to save the current configuration to your automation dashboard. You can retrieve saved reports any time by using **Select a Report**.

Export automation reports

You can export your automation usage and ROI data as CSV or PDF files to share with stakeholders.

Prerequisites

You must have configured your **Monthly AAP cost** and **Hourly rate** in the settings to ensure accurate ROI data in the exported report.

Procedure

1. Select a saved report or use the filters to customize the current view.
2. Select a **Duration** for the data you want to export.
3. Click **Export as CSV**: Generates a spreadsheet of the raw data.
4. Click **Save as PDF**: Generates a formatted summary report.

Related information
Create repeatable, shareable job templates to standardize automation runs

Summary of key automation data metrics

Automation dashboard displays a summary of the top and overview usage for your selected report. This includes the following data:

- **Successful jobs**: Number of job runs that completed without error in the selected period. Use the ratio between successful and failed jobs to track automation health and reliability over time.
- **Failed jobs**: Number of job runs that ended in failure in the selected period. Review failed jobs to fix playbooks, credentials, or inventory issues and improve success rates.
- **Hosts automated**: Number of hosts that executed at least one automation job in the selected period. Indicates how much of your inventory is actively automated and can help with license or capacity planning.
- **Hours of automation**: Sum of all job runtimes in the selected period. Reflects total automation workload and can inform capacity planning and resource allocation.
- **Number of times jobs were run**: Total number of job executions in the selected period, regardless of success or failure. Use this to understand automation volume, trends, and adoption over time.
- **Number of hosts jobs are running on**: Number of hosts that ran at least one job in the selected period. Complements run count by showing how broadly automation is applied across your inventory.
- **Top 5 projects**: Projects ranked by total job count in the selected period. Helps identify which projects are driving the most automation activity.
- **Top 5 users**: Users ranked by automation runs they triggered or that ran in their context in the selected period. Shows individual adoption and activity.

> **NOTE:**
> Scheduled jobs can affect these results, because they do not represent a real, logged-in user.

Calculate savings from automation

To calculate total savings, the cost and savings analysis compares manual labor costs against Ansible Automation Platform execution costs. The dashboard prorates the monthly subscription cost into daily reporting values.

Procedure

1. In the **Monthly AAP cost** field, enter the monthly cost of running the Ansible Automation Platform. This value includes license, labor, and infrastructure costs to run Ansible Automation Platform. It is used to calculate the automation savings.
2. In the **Hourly rate for manually running the job ($)** field, enter the hourly labor cost used to estimate what it would cost to run these jobs manually. This is used to calculate manual cost and savings.
3. Optional: Select **Time taken to create automation** to include the estimated time spent creating or authoring the automation (for example, writing playbooks, setting up jobs) before it could be run.

   Automation dashboard calculates the following data points based on your inputs:

   - **Cost of manual automation**: Total cost if all jobs were run manually.
   - **Cost of automated execution**: Total cost of running jobs on Ansible Automation Platform.
   - **Total savings/cost avoided**: Difference between manual and automated cost.
   - **Total hours saved/avoided**: Time saved by automation vs manual execution.
   - **Time taken to manually execute (min)**: Estimated manual execution time in minutes.
   - **Time taken to create automation (min)**: Estimated time spent creating or authoring the automation (for example, writing playbooks, setting up jobs) before it could be run. Included in cost when the switch above is on.

Result

Navigate to the Reports page to view the **Total savings** based on your updated cost configurations.

Inventory file variables for automation dashboard

The inventory variables required by the automation dashboard installation program are described in the following table:

Inventory variable	Description
`aap_auth_provider_name`	Natural language name shown on the login page. Default: Ansible Automation Platform
`aap_auth_provider_protocol`	Enter http or https
`aap_auth_provider_aap_version`	The version of Ansible Automation Platform that automation dashboard connects to. Enter the version number of your Ansible Automation Platform deployment (for example, `2.6`).

Inventory variable	Description
`aap_auth_provider_host`	Ansible Automation Platform IP or DNS name, with optional port
`aap_auth_provider_check_ssl`	Enforce TLS check or not
`aap_auth_provider_client_id`	Ansible Automation Platform OAuth2 application `client_id`. Required for SSO authentication.
`aap_auth_provider_client_secret`	Ansible Automation Platform OAuth2 application `client_secret`
`initial_sync_days`	Requests x number of days of old data, counting from "today"
`initial_sync_since`	Requests data from the specified date until "today"
`dashboard_update_secret`	Forces regeneration of autogenerated Podman secrets. Store the password for database access by using a Podman secret. Set `dashboard_update_secret` to true if you changed the `dashboard_pg_password` in inventory.
`nginx_disable_https`	Enables use of http instead of https for automation dashboard
`nginx_http_port`	Configures the HTTP port for automation dashboard
`nginx_https_port`	Configures the HTTPS port for automation dashboard
`dashboard_tls_cert`	The public TLS certificate for the automation dashboard. This file must include the server certificate, intermediate CA certificates, and the root CA certificate. The server certificate must appear at the beginning of the file.
`dashboard_tls_key`	The private TLS key for the automation dashboard. If not specified, the installation program generates a self-signed key.
`postgresql_admin_username`	Admin username to access PostgreSQL database
`postgresql_admin_password`	Admin password to access PostgreSQL database
`registry_username`	Username used to pull container images from `registry.redhat.io`. This variable is required when using an online installation program.
`registry_password`	Password used to pull container images from `registry.redhat.io`. This variable is required when using an online installation program.
`dashboard_pg_containerized`	Configures the installation program to install and configure the database as a container on the same host as automation dashboard. `True` is also the only supported value.
`dashboard_admin_password`	The password for the automation dashboard administrator user. The username is always `admin`.
`dashboard_pg_host`	The hostname or IP address of the PostgreSQL database. This host must be accessible from the installation program node on port 5432. If a public IP address blocks this port, you must use the internal IP address.

Inventory variable	Description
`dashboard_pg_username`	The database user for automation dashboard.
`dashboard_pg_password`	The password for the `dashboard_pg_username`.
`dashboard_pg_database`	The database schema name for automation dashboard.
`bundle_install`	Indicates the required container images are already included in the installation bundle (tar file).
`bundle_dir`	The directory where the installation bundle unpacks `+ /bundle` (for example: */home/<username>/ansible-automation-dashboard-containerized-setup/bundle*). The default value is relative to the current directory (PWD) and should work without modification.
`redis_mode`	Configures the Redis deployment mode. The required value is `standalone`.
`registry_url_aap_automation_dashboard`	Configures a custom container registry URL specifically for the automation dashboard images. Use this if you must pull dashboard images from a different registry than your database or Redis images.
`registry_ns_aap_automation_dashboard`	Configures a custom namespace specifically for the automation dashboard images.
`registry_username_aap_automation_dashboard`	The username for the custom registry defined in `registry_url_aap_automation_dashboard`. Required if the dashboard registry differs from the primary registry.
`registry_password_aap_automation_dashboard`	The password for the custom registry defined in `registry_url_aap_automation_dashboard`. Required if the dashboard registry differs from the primary registry.
`registry_tls_verify_aap_automation_dashboard`	Enforces or disables TLS certificate verification when pulling automation dashboard images from a custom registry. Default: `True`.

Cost and savings analysis metrics

The costs and savings analysis generates metrics that quantify the return on investment (ROI) derived from automation execution.

The costs and savings analysis generates the following metrics to quantify the return on investment (ROI) derived from automation execution.

Metric	Description
Cost of manual automation	Total cost if all jobs were run manually. Calculated as: (Manual time in minutes × Host executions) × Average labor cost per minute.

Metric	Description
Cost of automated execution	Total cost of running jobs on Ansible Automation Platform. Calculated as: (Running time in minutes × Prorated subscription cost per minute).
Total savings/cost avoided	The difference between manual and automated costs.
Total hours saved/ avoided	Time saved by automation compared to manual execution.
Time taken to manually execute (min)	Estimated manual execution time in minutes.
Time taken to create automation (min)	Estimated time spent creating or authoring the automation (for example, writing playbooks or setting up jobs). Included in cost calculations when enabled in settings.
Hourly rate for manually running the job ($)	The hourly labor cost used to estimate the expense of running jobs manually. Used to calculate manual cost and savings.
Monthly cost of running Ansible Automation Platform	Monthly cost of running the Ansible Automation Platform. This value includes license, labor and infrastructure costs to run Ansible Automation Platform. It is used to calculate the automation savings.

Manage analytics and data collection in Ansible Automation Platform

You can change how you participate in usability analytics and data collection from automation controller by opting out or changing your settings in the automation controller user interface.

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.

Related information
Configure Automation Analytics

Control data collection from automation controller

You can control how automation controller collects data from the **Settings > Automation Execution > System** menu.

Procedure

1. Log in to your automation controller.
2. From the navigation panel, select **Settings > Automation Execution > System**.
3. Select **Gather data for Automation Analytics** to enable automation controller to gather data on automation and send it to Automation Analytics.

Optimize request timeouts

Ansible Automation Platform on OpenShift Container Platform manages request timeouts through a synchronized, cascading architecture. This structure ensures that if a high-level service (the Gateway) reaches a timeout limit, all dependent internal processes terminate.

placeholder

Cascading client timeouts

Cascading timeouts ensure that if an outer layer times out, inner processes also terminate to prevent resource exhaustion. Set the primary timeout at the Gateway level to synchronize timeouts across applications.

Timeout mathematical relationships

Ansible Automation Platform maintains a downward mathematical cascade to ensure correct behavior and assist with debugging. Ansible Automation Platform applies the following logic to internal layers:

- The `client_request_timeout` serves as the primary value from which others are derived.

- The sum of the Envoy `request_timeout` and the gRPC authentication timeout (`gateway_grpc_auth_service_timeout`) is less than the `client_request_timeout` .
- The Nginx read timeout (`nginx_read_timeout`) is less than or equal to the Envoy `request_timeout` .
- The Python web server timeout (`python_webserver_timeout`) is less than or equal to the `nginx_read_timeout` .

Timeout grace periods

At the uWSGI layer, the `uwsgi_timeout_grace_period` allows the application to attempt a graceful shutdown. During this period, the application displays a traceback of the current stack position at the time of the timeout. If the process does not exit within the grace period, Ansible Automation Platform maintains forcefully terminates it.

Increase the OpenShift Route timeout

High-volume API operations can exceed the default 30-second OpenShift Route timeout. To resolve HTTP 504 or 503 errors, you must increase `client_request_timeout` in the `AnsibleAutomationPlatform` custom resource..

Before you begin

- Access to the Red Hat OpenShift Container Platform (RHOCP) web console.
- Update the Ansible Automation Platform 2.6 operator to the latest version to ensure configuration changes propagate correctly to the OpenShift Route.

Procedure

1. Log in to the OpenShift web console.
2. Navigate to **Installed Operators → Ansible Automation Platform → All Instances**.
3. Click your `AnsibleAutomationPlatform` instance.
4. Select the **YAML** tab. Under the spec: section, add the `route_annotations` to extend the timeout:

```
spec:
  route_annotations: |
    haproxy.router.openshift.io/timeout: 180s
```

5. Save the changes.

Result

Confirm that the `haproxy.router.openshift.io/timeout` annotation in the **Route** reflects the new value.

- Navigate to **Networking** → **Routes** in the OpenShift console.
  Select the route for your **Ansible Automation Platform** instance.
- Verify the **Annotations** section contains the updated timeout value.

Red Hat product documentation legal notices

GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If

such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions. "This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code. The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component,

or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions. All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law. No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work’s users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies. You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions. You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
- c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so. A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms. You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

- c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms. "Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination. You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies. You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients. Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents. A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom. If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License. Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License. The Free Software Foundation may publish revised and/ or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty. THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES

SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16. If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

```
    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <https://www.gnu.org/licenses/>.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
    <program>  Copyright (C) <year>  <name of author>
    This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.
```

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see link:https://www.gnu.org/licenses/.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read link:https://www.gnu.org/licenses/why-not-lgpl.html.

Apache license

Version 2.0, January 2004

http://www.apache.org/licenses/

Terms and Conditions for use, reproduction, and distribution

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, **"control"** means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or **"Your"**) shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, **"submitted"** means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the

Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as **"Not a Contribution."**

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

1. You must give any other recipients of the Work or Derivative Works a copy of this License; and
2. You must cause any modified files to carry prominent notices stating that You changed the files; and
3. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
4. If the Work includes a **"NOTICE"** text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications,

or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

www.redhat.com