



Administer

Ansible Automation Platform 2.6



May 12, 2026

Contents

1 Administer	11
Define automation target hosts in your inventory files.....	11
Define a collection of hosts with Smart Inventories	12
Smart Host Filters	13
Configure supported inventory plugin templates.....	15
Amazon Web Services EC2	16
Google Compute Engine	19
Microsoft Azure Resource Manager.....	20
VMware vCenter	22
VMWare ESXi.....	23
Red Hat Satellite 6	25
OpenStack	26
Red Hat Ansible Automation Platform	27
Create dynamic groups with constructed inventories	27
Debugging tips	28
Nested groups	29
Ansible facts.....	30
Filter on environment variables	30
Filter hosts by processor type.....	31
Filtering on group name and variables.....	32
Example	32
Generate dynamic data from external sources with inventory plugins	33
Add permissions to inventories.....	34
Remove a permission.....	35
Add groups to inventories	35
Add groups within groups	36
View or edit inventory groups	37
Add hosts to inventory groups.....	37
Add a source to an inventory.....	38
Source from a project's source control management.....	41
Source an inventory from Amazon Web Services EC2	42

Source an inventory from Google Compute Engine	43
Source an inventory from Microsoft Azure resource manager.....	43
Source an inventory from VMware vCenter	44
Source an inventory from VMware ESXi	45
Source an inventory from Red Hat Satellite 6.....	45
Source an inventory from Red Hat Lightspeed.....	46
Source an inventory from OpenStack	47
Source an inventory from Red Hat Virtualization	47
Source an inventory from Red Hat Ansible Automation Platform	48
Source an inventory from Terraform State	49
Source an inventory from OpenShift Virtualization	50
Source an inventory from a custom inventory plugin.....	51
Export old inventory scripts.....	52
Run ad hoc commands against a host in an inventory	53
Configure notifications for the source	55
View completed jobs	55
Import your inventory file from source control	56
Inventory file fields for source control	56
Supported file syntax	57
Define where automation runs with host and node groupings	59
How Ansible Automation Platform supports host and node groupings	59
Add new hosts as automation targets	59
Create a host	60
View the host details.....	60
Determine where automation runs with instance groups	61
Instance groups	62
Create an instance group	63
Associate instances to an instance group.....	64
View jobs associated with an instance group.....	64
Control where an automation job runs	65
Define instance groups in the inventory file	66
Define instance groups	67
Configure instance groups from the API.....	68
Instance group policies.....	69
Job slice considerations.....	69

Job runtime behavior	71
Pin instances manually to specific instance groups.....	71
Capacity limits for instance groups	72
Deprovision instance groups.....	73
Control where automation runs with container groups.....	74
Create a container group.....	75
Create a service account in OpenShift Container Platform or Kubernetes	76
Customize the pod specification	78
Verify the container group functions	79
Container capacity limits	81
Scale automation across your infrastructure with automation mesh	82
About automation mesh	82
Control and execution planes	83
Control plane	83
Execution plane	84
Peers	84
Define automation mesh node types.....	84
Hybrid nodes.....	85
Control nodes	85
Execution nodes.....	85
Hop nodes	86
Peer connections	87
Scale with automation mesh in an operator environment	87
Prerequisites.....	88
Set up VMs for use in mesh	88
Add execution or hop nodes to scale your infrastructure	90
Create an instance group	96
Associate instances to an instance group.....	97
Run jobs on execution nodes	97
Set up mesh ingress for environments that forbid inbound connections	98
Create a pull secret to run the default execution environment on remote nodes	100
Use custom signed certificates in managed cloud and operator environments.....	101
Remove instances	102
Upgrade receptors.....	103

Administer

View details about mesh nodes with topology viewer.....	104
Access the topology viewer.....	104
Scale with automation mesh in a containerized or VM environment.....	105
Edit the inventory file	105
For an RPM installation	106
For online installations.....	106
Run the setup script.....	107
Import a CA certificate	107
Design patterns for mesh	108
Multiple hybrid nodes inventory file example	109
Single node control plane with single execution node	110
Minimum resilient configuration	112
Segregated local and remote execution configuration	114
Multi-hopped execution node	116
Outbound only connections to controller nodes	118
Back up and restore your containerized deployment.....	119
Back up containerized Ansible Automation Platform.....	120
Restore containerized Ansible Automation Platform.....	122
Back up and restore in an OpenShift environment.....	124
About backup and recovery	125
Backup recommendations.....	125
Back up your Ansible Automation Platform deployment	125
Back up your Ansible Automation Platform deployment	126
Define custom backup Persistent Volume Claims.....	127
Custom backup configurations for specific components.....	131
Resolve naming conflicts when deploying an Automation Controller custom resource	133
Restore your Ansible Automation Platform deployment	133
Restore Ansible Automation Platform.....	133
Restore from a persistent volume claim (PVC)	135
Restore from an external database.....	137
Back up and restore your RPM deployment	138
Backup and restore overview	139
Backup and restore playbooks.....	140

Administer

Considerations for backup and restore	141
Back up your Ansible Automation Platform instance	142
Backup and restore in clustered environments	143
Restore to a different cluster	143
Define, create, and build execution environments	144
Build a definition file	145
Definition dependencies	146
Galaxy	147
Python.....	147
System	148
Images.....	149
Disconnected environment customizations.....	149
Execution environment setup reference.....	150
Execution environment definition example	150
Configuration options	151
Additional_build_files	152
Additional_build_steps	153
build_arg_defaults	153
Dependencies	154
Images.....	158
Image verification.....	158
Options	159
Version	160
Set up your container repository	160
Add a README to your container repository	161
Provide access to your automation execution environments.....	161
Tag container images	162
Create a credential	162
Pull execution environments for use in automation hub.....	163
Tag execution environments.....	164
Push an execution environment to private automation hub	164
Pull and sync images from automation hub to your local system.....	165
Pull an image.....	166
Sync images from a container registry	166

Administer

Secure your automation with container signing	167
Deploy your system for container signing.....	167
Add and sign an execution environment	169
Push a signed execution environment from your local system.....	170
Confirm signatures are on your local environment.....	171
Configure a client to verify signatures.....	171
Manage your organization's automation content	173
Sync automation content from remote repositories	174
Red Hat Certified, validated, and Ansible Galaxy content in automation hub	175
Configure automation hub tokens	176
Create the offline token in automation hub	176
Refresh the offline token	177
Configure the rh-certified remote repository	177
Configure the private automation hub community remote	178
Configure proxy settings	179
Create a requirements file	180
Configure content signing on private automation hub	180
Download signature public keys	182
Configure Ansible-Galaxy CLI to verify collections	183
Manage collection access and permissions with namespaces.....	184
Create a content development team	184
Create a namespace.....	185
Edit namespace resources.....	186
Upload collections to a namespace	187
Review your namespace import logs	188
Delete a collection	189
Delete a namespace	190
Upload, group, and approve your organization's content collections	190
Approve collections	191
Reject collections uploaded for review.....	191
Repository management with automation hub	192
Types of repositories in automation hub.....	192
Approve content for custom repositories in automation hub.....	192
Manage access to custom content with role based access control.....	193

Administer

Create a custom repository.....	193
Configure access to a custom automation hub repository	194
Add automation content to an automation hub repository	195
Manage automation hub repository versions	195
Create a remote configuration in automation hub	196
Assign access to a remote configuration	197
Synchronize repositories in automation hub	198
Export automation content from automation hub	198
Import automation content to automation hub	199
Manage containers in your private automation hub	199
Populate your private automation hub container registry.....	200
Pull execution environments for use in automation hub.....	201
Tag container images	202
Pull and sync images from automation hub to your local system.....	203
Pull an image.....	203
Sync images from a container registry	204
Create a credential	205
Remote registry team permissions.....	205
Add a README to your container repository	206
Trigger automation from events with Event-Driven Ansible	206
Organize rulebooks for event-driven automation	207
Set up a new project	208
Projects list view	210
Edit a project.....	210
Delete a project.....	211
Create runtime environments for event-driven automation	211
Supported event sources and filters.....	212
Build a custom decision environment for Event-Driven Ansible	213
Set up a new decision environment	216
Define rules that trigger automation from events	218
Set up a rulebook activation.....	219
Rulebook activation list view	222
View activation output	222

Manage active event-driven automation integrations.....	223
Enable and disable rulebook activations	223
Restart rulebook activations	223
Edit a rulebook activation	224
Delete rulebook activations.....	224
Reuse event-driven automation configurations	225
Duplicate a rulebook activation	225
Activate webhook-based automation in Openshift.....	226
Testing with OpenShift (Kubernetes)	229
Audit pipeline-triggered automation activity	229
View rule audit details.....	229
View rule audit events	230
View rule audit actions.....	230
Troubleshoot failed event-driven automation triggers	230
Event-Driven Ansible log filtering	231
Use log filtering for troubleshooting.....	232
Resolve rulebook activations stuck in pending state	233
Fix rulebook activations stuck in a restart loop	233
Resolve event processing failures in rulebook activations	234
Troubleshoot actions that fail to trigger after receiving events.....	234
Troubleshoot event streams that fail to send events	235
Respond to events from external systems.....	236
Event streams	236
Create an event stream credential.....	238
Create an event stream	239
HTTP headers.....	241
Configuring HTTP headers securely for event streams.....	241
Static Unique Universal Identifiers (UUIDs) for event streams	242
Update an event stream with a static UUID (API Method)	243
Configure your remote system to send events.....	245
Verify your event streams work	246
Replace sources and attach event streams to activations	247
Filter events before triggering automation	252
Author event filters.....	253

Administer

Use awx-manage to access automation controller information.....	254
Inventory Import.....	254
Cleanup of old data	255
Cluster management.....	255
Analytics gathering	255
Red Hat product documentation legal notices	256
GNU GENERAL PUBLIC LICENSE	257
Apache license	268

1 Administer

Define automation target hosts in your inventory files

Ansible Automation Platform works against a list of managed nodes or hosts in your infrastructure that are logically organized, using an inventory file. The installer inventory can specify your installation scenario and describe host deployments to Ansible.

An inventory file enables Ansible to manage a large number of hosts with a single command. Inventories also help you automate more efficiently by reducing the number of command line options you have to specify. Inventories are divided into groups and these groups contain the hosts. Groups can be sourced manually, by entering host names into automation controller, or from one of its supported cloud providers.

NOTE:

If you have a custom dynamic inventory script, or a cloud provider that is not yet supported natively in automation controller, you can also import that into automation controller.

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

The **Inventory details** page includes:

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

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

Related information

[Import your inventory file from source control](#)

Define a collection of hosts with Smart Inventories

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

You can identify a Smart Inventory by the `KIND=smart` variable.

You can define a Smart Inventory by using the same method being used with Search.

`InventorySource` is directly associated with an Inventory.

NOTE:

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

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

- Set `kind` to `smart` for Smart Inventories.
- Set `host_filter` AND `kind` to `smart` for Smart Inventories.

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

NOTE:

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

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

You can view inventories without being editable:

- Names of Host and Group created as a result of an inventory source synchronization.

- You cannot move or edit Group records.

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

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

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

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

In some situations, you can change the following:

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

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

Smart Host Filters

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

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

For example:

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

The `host_filter` parameter permits:

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

Examples:

```
/api/v2/hosts/?host_filter=name=localhost
/api/v2/hosts/?host_filter=ansible_facts__ansible_date_time__weekday_number="3"
/api/v2/hosts/?host_filter=ansible_facts__ansible_processor[]="GenuineIntel"
/api/v2/hosts/?host_filter=ansible_facts__ansible_lo__ipv6[]__scope="host"
/api/v2/hosts/?host_filter=ansible_facts__ansible_processor_vcpus=8
/api/v2/hosts/?host_filter=ansible_facts__ansible_env__PYTHONUNBUFFERED="true"
/api/v2/hosts/?host_filter=(name=localhost or name=database) and (groups__name=east
or groups__name="west coast") and ansible_facts__an
```

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

Group search has the following format:

```
groups.name:groupA
```

Fact search has the following format:

```
ansible_facts.ansible_fips:false
```

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

```
host_filter=name=my_host
```

NOTE:

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

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

Configure supported inventory plugin templates

After upgrade to 4.x, existing configurations are migrated to the new format that produces an inventory output compatible with earlier versions. Use the following templates to aid in migrating your inventories to the new style inventory plugin output.

- Amazon Web Services EC2
- Google Compute Engine
- Microsoft Azure Resource Manager
- VMware vCenter
- VMware ESXi
- Red Hat Satellite 6
- OpenStack
- Red Hat Ansible Automation Platform

Related information

[Amazon Web Services EC2](#)

[Google Compute Engine](#)

[Microsoft Azure Resource Manager](#)

[VMware vCenter](#)

[VMware ESXi](#)

[Red Hat Satellite 6](#)

[OpenStack](#)

[Red Hat Ansible Automation Platform](#)

Amazon Web Services EC2

Use the following reference to configure an AWS EC2-sourced inventory.

```
compose:
  ansible_host: public_ip_address
  ec2_account_id: owner_id
  ec2_ami_launch_index: ami_launch_index | string
  ec2_architecture: architecture
  ec2_block_devices: dict(block_device_mappings | map(attribute='device_name') |
list | zip(block_device_mappings | map(attribute='ebs.volume_id') | list))
  ec2_client_token: client_token
  ec2_dns_name: public_dns_name
  ec2_ebs_optimized: ebs_optimized
  ec2_eventsSet: events | default("")
  ec2_group_name: placement.group_name
  ec2_hypervisor: hypervisor
  ec2_id: instance_id
  ec2_image_id: image_id
  ec2_instance_profile: iam_instance_profile | default("")
  ec2_instance_type: instance_type
  ec2_ip_address: public_ip_address
  ec2_kernel: kernel_id | default("")
  ec2_key_name: key_name
  ec2_launch_time: launch_time | regex_replace(" ", "T") | regex_replace("(\\+)(\\d\\d):(\\d)(\\d)$", ".\\g<2>\\g<3>Z")
  ec2_monitored: monitoring.state in ['enabled', 'pending']
  ec2_monitoring_state: monitoring.state
  ec2_persistent: persistent | default(false)
  ec2_placement: placement.availability_zone
  ec2_platform: platform | default("")
  ec2_private_dns_name: private_dns_name
  ec2_private_ip_address: private_ip_address
  ec2_public_dns_name: public_dns_name
  ec2_ramdisk: ramdisk_id | default("")
  ec2_reason: state_transition_reason
  ec2_region: placement.region
```

```

ec2_requester_id: requester_id | default("")
ec2_root_device_name: root_device_name
ec2_root_device_type: root_device_type
ec2_security_group_ids: security_groups | map(attribute='group_id') | list |
join(',')
ec2_security_group_names: security_groups | map(attribute='group_name') | list |
join(',')
ec2_sourceDestCheck: source_dest_check | default(false) | lower | string
ec2_spot_instance_request_id: spot_instance_request_id | default("")
ec2_state: state.name
ec2_state_code: state.code
ec2_state_reason: state_reason.message if state_reason is defined else ""
ec2_subnet_id: subnet_id | default("")
ec2_tag_Name: tags.Name
ec2_virtualization_type: virtualization_type
ec2_vpc_id: vpc_id | default("")
filters:
  instance-state-name:
    - running
groups:
  ec2: true
hostnames:
  - network-interface.addresses.association.public-ip
  - dns-name
  - private-dns-name
keyed_groups:
  - key: image_id | regex_replace("[^A-Za-z0-9\\_]", "_")
    parent_group: images
    prefix: ''
    separator: ''
  - key: placement.availability_zone
    parent_group: zones
    prefix: ''
    separator: ''
  - key: ec2_account_id | regex_replace("[^A-Za-z0-9\\_]", "_")
    parent_group: accounts
    prefix: ''

```

```

    separator: ''
  - key: ec2_state | regex_replace("[^A-Za-z0-9\\_]", "_")
    parent_group: instance_states
    prefix: instance_state
  - key: platform | default("undefined") | regex_replace("[^A-Za-z0-9\\_]", "_")
    parent_group: platforms
    prefix: platform
  - key: instance_type | regex_replace("[^A-Za-z0-9\\_]", "_")
    parent_group: types
    prefix: type
  - key: key_name | regex_replace("[^A-Za-z0-9\\_]", "_")
    parent_group: keys
    prefix: key
  - key: placement.region
    parent_group: regions
    prefix: ''
    separator: ''
  - key: security_groups | map(attribute="group_name") | map("regex_replace", "[^A-
Za-z0-9\\_]", "_") | list
    parent_group: security_groups
    prefix: security_group
  - key: dict(tags.keys() | map("regex_replace", "[^A-Za-z0-9\\_]", "_") | list |
zip(tags.values()
    | map("regex_replace", "[^A-Za-z0-9\\_]", "_") | list))
    parent_group: tags
    prefix: tag
  - key: tags.keys() | map("regex_replace", "[^A-Za-z0-9\\_]", "_") | list
    parent_group: tags
    prefix: tag
  - key: vpc_id | regex_replace("[^A-Za-z0-9\\_]", "_")
    parent_group: vpcs
    prefix: vpc_id
  - key: placement.availability_zone
    parent_group: '{{ placement.region }}'
    prefix: ''
    separator: ''
plugin: amazon.aws.aws_ec2

```

```
use_contrib_script_compatible_sanitization: true
```

Google Compute Engine

This inventory plugin dynamically generates inventory from Google Compute Engine (GCE) instances.

```
auth_kind: serviceaccount
compose:
  ansible_ssh_host: networkInterfaces[0].accessConfigs[0].natIP |
default(networkInterfaces[0].networkIP)
  gce_description: description if description else None
  gce_id: id
  gce_image: image
  gce_machine_type: machineType
  gce_metadata: metadata.get("items", []) | items2dict(key_name="key",
value_name="value")
  gce_name: name
  gce_network: networkInterfaces[0].network.name
  gce_private_ip: networkInterfaces[0].networkIP
  gce_public_ip: networkInterfaces[0].accessConfigs[0].natIP | default(None)
  gce_status: status
  gce_subnetwork: networkInterfaces[0].subnetwork.name
  gce_tags: tags.get("items", [])
  gce_zone: zone
hostnames:
- name
- public_ip
- private_ip
keyed_groups:
- key: gce_subnetwork
  prefix: network
- key: gce_private_ip
  prefix: ''
  separator: ''
- key: gce_public_ip
```

```
    prefix: ''
    separator: ''
  - key: machineType
    prefix: ''
    separator: ''
  - key: zone
    prefix: ''
    separator: ''
  - key: gce_tags
    prefix: tag
  - key: status | lower
    prefix: status
  - key: image
    prefix: ''
    separator: ''
plugin: google.cloud.gcp_compute
retrieve_image_info: true
use_contrib_script_compatible_sanitization: true
```

Microsoft Azure Resource Manager

The Microsoft Azure inventory plugin integrates with Microsoft Azure to pull information about virtual machines and other resources for use in automation tasks.

```

conditional_groups:
  azure: true
default_host_filters: []
fail_on_template_errors: false
hostvar_expressions:
  computer_name: name
  private_ip: private_ipv4_addresses[0] if private_ipv4_addresses else None
  provisioning_state: provisioning_state | title
  public_ip: public_ipv4_addresses[0] if public_ipv4_addresses else None
  public_ip_id: public_ip_id if public_ip_id is defined else None
  public_ip_name: public_ip_name if public_ip_name is defined else None
  tags: tags if tags else None
  type: resource_type
keyed_groups:
- key: location
  prefix: ''
  separator: ''
- key: tags.keys() | list if tags else []
  prefix: ''
  separator: ''
- key: security_group
  prefix: ''
  separator: ''
- key: resource_group
  prefix: ''
  separator: ''
- key: os_disk.operating_system_type
  prefix: ''
  separator: ''
- key: dict(tags.keys() | map("regex_replace", "^(.*)$", "\1_") | list |
zip(tags.values() | list)) if tags else []
  prefix: ''
  separator: ''
plain_host_names: true
plugin: azure.azcollection.azure_rm
use_contrib_script_compatible_sanitization: true

```

VMware vCenter

Configure and use VMware vCenter as a platform for hosting automation controller virtual machines (VMs).

```
compose:
  ansible_host: guest.ipAddress
  ansible_ssh_host: guest.ipAddress
  ansible_uuid: 99999999 | random | to_uuid
  availablefield: availableField
  configissue: configIssue
  configstatus: configStatus
  customvalue: customValue
  effectiverole: effectiveRole
  guestheartbeatstatus: guestHeartbeatStatus
  layoutex: layoutEx
  overallstatus: overallStatus
  parentvapp: parentVApp
  recenttask: recentTask
  resourcepool: resourcePool
  rootsnapshot: rootSnapshot
  triggeredalarmstate: triggeredAlarmState
filters:
- runtime.powerState == "poweredOn"
keyed_groups:
- key: config.guestId
  prefix: ''
  separator: ''
- key: '"templates" if config.template else "guests"'
  prefix: ''
  separator: ''
plugin: community.vmware.vmware_vm_inventory
properties:
- availableField
- configIssue
- configStatus
```

```

- customValue
- datastore
- effectiveRole
- guestHeartbeatStatus
- layout
- layoutEx
- name
- network
- overallStatus
- parentVApp
- permission
- recentTask
- resourcePool
- rootSnapshot
- snapshot
- triggeredAlarmState
- value
- capability
- config
- guest
- runtime
- storage
- summary
strict: false
with_nested_properties: true

```

VMWare ESXi

This example inventory configuration file demonstrates how to use the `vmware.vmware.vms` inventory plugin to gather virtual machine information from a VMWare ESXi host.

```

compose:
  ansible_host: guest.ipAddress
  ansible_ssh_host: guest.ipAddress
  ansible_uuid: 99999999 | random | to_uuid

```

```

    availablefield: availableField
    configissue: configIssue
    configstatus: configStatus
    customvalue: customValue
    effectiverole: effectiveRole
    guestheartbeatstatus: guestHeartbeatStatus
    layoutex: layoutEx
    overallstatus: overallStatus
    parentvapp: parentVApp
    recenttask: recentTask
    resourcepool: resourcePool
    rootsnapshot: rootSnapshot
    triggeredalarmstate: triggeredAlarmState
  filters:
    - runtime.powerState == "poweredOn"
  keyed_groups:
    - key: config.guestId
      prefix: ""
      separator: ""
    - key: '"templates" if config.template else "guests"'
      prefix: ""
      separator: ""
  plugin: vmware.vmware.vms
  properties:
    - availableField
    - configIssue
    - configStatus
    - customValue
    - datastore
    - effectiveRole
    - guestHeartbeatStatus
    - layout
    - layoutEx
    - name
    - network
    - overallStatus

```

```
- parentVApp
- permission
- recentTask
- resourcePool
- rootSnapshot
- snapshot
- triggeredAlarmState
- value
- capability
- config
- guest
- runtime
- storage
- summary
strict: false
flatten_nested_properties: true
```

Red Hat Satellite 6

You can use automation controller to manage hosts registered to Red Hat Satellite 6 by using the Foreman dynamic inventory plugin.

```

group_prefix: foreman_

keyed_groups:
- key: foreman['environment_name'] | lower | regex_replace(' ', '') |
  regex_replace('[^A-Za-z0-9_]', '_') | regex_replace('none', '')

  prefix: foreman_environment_

  separator: ''

- key: foreman['location_name'] | lower | regex_replace(' ', '') |
  regex_replace('[^A-Za-z0-9_]', '_')

  prefix: foreman_location_

  separator: ''

- key: foreman['organization_name'] | lower | regex_replace(' ', '') |
  regex_replace('[^A-Za-z0-9_]', '_')

  prefix: foreman_organization_

  separator: ''

- key: foreman['content_facet_attributes']['lifecycle_environment_name'] | lower |
  regex_replace(' ', '') | regex_replace('[^A-Za-z0-9_]', '_')

  prefix: foreman_lifecycle_environment_

  separator: ''

- key: foreman['content_facet_attributes']['content_view_name'] | lower |
  regex_replace(' ', '') | regex_replace('[^A-Za-z0-9_]', '_')

  prefix: foreman_content_view_

  separator: ''

legacy_hostvars: true

plugin: theforeman.foreman.foreman

validate_certs: false

want_facts: true

want_hostcollections: false

want_params: true

```

OpenStack

The OpenStack inventory plugin integrates with OpenStack clouds to dynamically generate Ansible inventories based on the current state of the cloud resources.

```
expand_hostvars: true
fail_on_errors: true
inventory_hostname: uuid
plugin: openstack.cloud.openstack
```

Red Hat Ansible Automation Platform

Learn about the configuration options available in a automation controller Red Hat Ansible Automation Platform inventory

```
include_metadata: true
inventory_id: <inventory_id or url_quoted_named_url>
plugin: awx.awx.tower
validate_certs: <true or false>
```

Create dynamic groups with constructed inventories

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

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

The key factors of a constructed inventory are:

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

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

You can construct groups based on these host properties:

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

- other

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

Debugging tips

When using constructed inventory scripts, you might need to debug your Jinja2 templates.

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

When encountering errors, increase verbosity to get more details.

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

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

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

Example

```
source_vars:

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

limit: ``
```

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

Nested groups

You can create nested groups in your inventory to organize hosts and apply variables at different levels.

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

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

The following example inventory file, `nested.yml` is in YAML format:

```
all:
  children:
    groupA:
      vars:
        filter_var: filter_val
      children:
        groupB:
          hosts:
            host1: {}
    ungrouped:
      hosts:
        host2: {}
```

Because `host1` is in `groupB`, it is also in `groupA`.

Filter on nested group names

Use the following YAML format to filter on nested group names:

```
`source_vars`:

plugin: constructed

`limit`: `groupA`
```

Filter on nested group property

Use the following YAML format to filter on a group variable, even if the host is indirectly a member of that group.

In the inventory content, note that `host2` is not expected to have the variable `filter_var` defined, because it is not in any of the groups. Because `strict: true` is used, use a default value so that hosts without that variable are defined. Using this, `host2`, returns `false` from the expression, instead of producing an error. `host1` inherits the variable from its groups, and is returned.

```
source_vars:

plugin: constructed

strict: true

groups:
  filter_var_is_filter_val: filter_var | default("") == "filter_val"

limit: filter_var_is_filter_val
```

Ansible facts

You can create an inventory that uses Ansible facts to populate host variables.

To create an inventory with Ansible facts, you must run a playbook against the inventory that has the setting `gather_facts: true`. The facts differ system-to-system. The following examples are not intended to address all known scenarios.

Filter on environment variables

You can filter hosts in an inventory by using the `ansible_env` variable in a constructed inventory plugin.

```
source_vars:

plugin: constructed
strict: true
groups:
  hosts_using_xterm: ansible_env.TERM == "xterm"

limit: hosts_using_xterm
```

Filter hosts by processor type

You can create a constructed inventory that filters hosts by CPU type by using the `ansible_processor` fact.

The following example involves filtering hosts by processor type (Intel) using the YAML format:

```
source_vars:

plugin: constructed
strict: true
groups:
  intel_hosts: "GenuineIntel" in ansible_processor

limit: intel_hosts
```

NOTE:

Hosts in constructed inventories are not counted against your license allotment because they are referencing the original inventory host. Additionally, hosts that are disabled in the original inventories are not included in the constructed inventory.

An inventory update run using `ansible-inventory` creates the constructed inventory contents.

This is always configured to update-on-launch before a job, but you can still select a cache timeout value in case this takes too long.

When creating a constructed inventory, the API ensures that it always has one inventory source associated with it. All inventory updates have an associated inventory source, and the fields needed for constructed inventory (`source_vars` and `limit`) are fields already present on the inventory source model.

Filtering on group name and variables

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

There are two approaches to executing this filter:

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

Example

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

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

```
[account_1234]
host1
host2 state=shutdown

[account_4321]
host3
host4 state=shutdown

[account_1234:vars]
account_alias=product_dev

[account_4321:vars]
account_alias=sustaining
```

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

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

```
source_vars :
```

```

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

```

```
limit: is_shutdown:&product_dev
```

This constructed inventory input creates a group for both categories and uses the `limit` (host pattern) to only return hosts that are in the intersection of those two groups.

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

2. Construct 1 group, limit to group:

```
source_vars:
```

```

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

```

```
limit: shutdown_in_product_dev
```

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

Generate dynamic data from external sources with inventory plugins

Automation controller uses inventory plugins to dynamically generate inventory data from external sources. These plugins enable automation controller to integrate with cloud providers, virtualization platforms, and other external systems to retrieve up-to-date information about managed nodes.

Inventory updates use dynamically-generated YAML files which are parsed by their inventory plugin. In automation controller, you can provide the inventory plugin configuration directly to automation controller using the inventory source `source_vars` for the following inventory sources:

- [Amazon Web Services EC2](#)

- [Google Compute Engine](#)
- [Microsoft Azure Resource Manager](#)
- [VMware vCenter](#)
- [VMWare ESXI](#)
- [Red Hat Satellite 6](#)
- [Red Hat Lightspeed](#)
- [OpenStack](#)
- [Red Hat Virtualization](#)
- [Red Hat Ansible Automation Platform](#)
- [Terraform State](#)
- [OpenShift Virtualization](#)

Newly created configurations for inventory sources contain the default plugin configuration values. If you want your newly created inventory sources to match the output of a legacy source, you must apply a specific set of configuration values for that source. To ensure backward compatibility, automation controller uses "templates" for each of these sources to force the output of inventory plugins into the legacy format.

For more information about sources and their templates, see [Supported inventory plugin templates](#).

`source_vars` that contain `plugin: xxx.yyy.zzz` as a top-level key are replaced with the fully-qualified inventory plugin name at runtime based on the `InventorySource` source.

For example, if you select `ec2` for the `InventorySource` then, at run time, the plugin is set to `amazon.aws.aws_ec2`.

Add permissions to inventories

Use the following procedure to add permissions to inventories:

Procedure

1. From the navigation panel, select **Automation Execution > Infrastructure > Inventories**.
2. Select a template, and in the **User Access** or **Team Access** tab, click **Add roles**.
3. Select a user or team to add and click **Next**.
4. Select the checkbox next to a name to add one or more users or teams from the list as members.
5. Click **Next**.

6. Select the roles you want the selected users or teams to have. Different resources have different options available.
7. Click **Finish** to apply the roles to the selected users or teams and to add them as members. The updated roles assigned for each user and team are displayed.

Remove a permission

Learn how to remove specific permissions from a user associated with a resource. Disassociating a role restricts a user's access to functionalities or data they no longer need.

- To remove roles for a particular user, click the **X** icon next to its resource. This launches a confirmation window, asking you to confirm the disassociation.

Add groups to inventories

You can add groups to standard inventories in automation controller to organize hosts.

Inventories are divided into groups, which can contain hosts and other groups. Groups are only applicable to standard inventories and are not a configurable directly through a Smart Inventory. You can associate an existing group through hosts that are used with standard inventories.

The following actions are available for standard inventories:

- Create a new Group
- Create a new Host
- Run a command on the selected Inventory
- Edit Inventory properties
- View activity streams for Groups and Hosts
- Obtain help building your Inventory

NOTE:

Inventory sources are not associated with groups. Spawned groups are top-level and can still have child groups. All of these spawned groups can have hosts.

Procedure

1. From the navigation panel, select **Automation Execution > Infrastructure > Inventories**.
2. Select the Inventory name you want to add groups to.
3. In the Inventory **Details** page, select the **Groups** tab.

4. Click **Create group**.
5. Enter the appropriate details:
 - **Name**: Required
 - Optional: **Description**: Enter a description as appropriate.
 - Optional: **Variables**: Enter definitions and values to be applied to all hosts in this group. Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two.
6. Click **Create group**.

Result

When you have added a group to a template, the Group **Details** page is displayed.

Add groups within groups

You can create a hierarchy of groups by adding groups within groups. This allows you to create nested groupings of hosts for easier management and organization.

When you have added a group to a template, the Group **Details** page is displayed.

Procedure

1. Select the **Related Groups** tab.
2. Click **Add existing group** to add a group that already exists in your configuration or **Create group** to create a new group.
3. If creating a new group, enter the appropriate details into the required and optional fields:
 - **Name** (required):
 - Optional: **Description**: Enter a description as appropriate.
 - Optional: **Variables**: Enter definitions and values to be applied to all hosts in this group. Enter variables using either JSON or YAML syntax. Use the radio button to toggle between the two.
4. Click **Create group**.
5. The **Create group** window closes and the newly created group is displayed as an entry in the list of groups associated with the group that it was created for.

Next steps

If you select to add an existing group, available groups appear in a separate selection window. When you select a group, it is displayed in the list of groups associated with the group.

- To configure additional groups and hosts under the subgroup, click the name of the subgroup from the list of groups and repeat the steps listed before.

View or edit inventory groups

The groups list view displays all your inventory groups, or you can filter it to only display the root groups. An inventory group is considered a root group if it is not a subset of another group.

You can delete a subgroup without concern for dependencies, because automation controller looks for dependencies such as child groups or hosts. If any exist, a confirmation window displays for you to select whether to delete the root group and all of its subgroups and hosts; or to promote the subgroups so they become the top-level inventory groups, along with their hosts.

Add hosts to inventory groups

You can add hosts to an inventory either by creating new hosts or by associating existing hosts with the inventory.

You can configure hosts for the inventory and for groups and groups within groups.

Procedure

1. From the navigation panel, select **Automation Execution > Infrastructure > Inventories**.
2. Select the inventory name you want to add groups to.
3. In the Inventory **Details** page, select the **Hosts** tab.
4. Click **Create host**.
5. Select whether to add a host that already exists in your configuration or create a new host.
6. If creating a new host, set the toggle to **On** to include this host while running jobs.
7. Enter the appropriate details:
 - **Name** (required):
 - Optional: **Description**: Enter a description as appropriate.
 - Optional: **Variables**: Enter definitions and values to be applied to all hosts in this group, as in the following example:

```
{  
  ansible_user : <username to ssh into>  
  ansible_ssh_pass : <password for the username>  
  ansible_become_pass: <password for becoming the root>  
}
```

Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two.

8. Click **Create host**.
9. The **Create host** window closes and the newly created host is displayed in the list of hosts associated with the group that it was created for.

If you select to add an existing host, available hosts appear in a separate selection window.

When you select a host, it is displayed in the list of hosts associated with the group.

10. You can disassociate a host from this screen by selecting the host and clicking the **X** icon.

NOTE:

You can also run ad hoc commands from this screen. For more information, see [Running Ad Hoc commands](#).

11. To configure additional groups for the host, click the name of the host from the list of hosts.
This opens the **Details** tab of the selected host.
12. Select the **Groups** tab to configure groups for the host.
13. Click **Associate groups** to associate the host with an existing group. Available groups appear in a separate selection window.
14. Select the groups to associate with the host and click **Confirm**.
When a group is associated, it is displayed in the list of groups associated with the host.
15. If you used a host to run a job, you can view details about those jobs in the **Completed Jobs** tab of the host.
16. Click **Expanded** to view details about each job.

NOTE:

You can create hosts in bulk by using the newly added endpoint in the API, `/api/v2/bulk/host_create`. This endpoint accepts JSON and you can specify the target inventory and a list of hosts to add to the inventory. These hosts must be unique within the inventory. Either all hosts are added, or an error is returned indicating why the operation was not able to complete. Use the **OPTIONS** request to return the relevant schema.

Add a source to an inventory

Use the following procedure to add a source to an inventory. When you add a source to an inventory, the system creates a new group for that source.

Inventory sources are not associated with groups. Spawned groups are top-level and can still have child groups. All of these spawned groups can have hosts. Adding a source to an inventory only applies to standard inventories.

Procedure

1. From the navigation panel, select **Automation Execution > Infrastructure > Inventories**.
2. Select the inventory name you want to add a source to.
3. In the Inventory **Details** page, select the **Sources** tab.
4. Click **Create source**.
5. Enter the appropriate details:
 - **Name** (required):
 - Optional: **Description**: Enter a description as appropriate.
 - Optional: **Execution Environment**: Click the  icon or enter the name of the execution environment with which you want to run your inventory imports.
 - **Source**: Choose a source for your inventory.
6. When the information for your chosen inventory sources is complete, you can optionally specify other common parameters, such as verbosity, host filters, and variables.
7. Use the **Verbosity** menu to select the level of output on any inventory source's update jobs.
8. Use the **Host filter** field to specify only matching host names to be imported into automation controller.
9. In the **Enabled variable** field, specify that automation controller retrieves the enabled state from the dictionary of host variables. You can specify the enabled variable by using dot notation as 'car.key', in which case the lookup searches nested dictionaries, equal to:


```
from_dict.get('car', {}).get('key', default)
```

The **Enabled value** field is ignored unless you set the **Enabled variable** field. If the enabled variable matches this value, the host is enabled on import.
10. If you specified a dictionary of host variables in the **Enabled variable** field, you can give a value to enable on import. For example, for `enabled_var='status.power_state'` and `'enabled_value='powered_on'` in the following host variables, the host is marked enabled :

```
{
  "status": {
    "power_state": "powered_on",
    "created": "2020-08-04T18:13:04+00:00",
    "healthy": true
  },
  "name": "foobar",
  "ip_address": "192.168.2.1"
}
```

If `power_state` is any value other than `powered_on`, then the host is disabled when imported into automation controller. If the key is not found, then the host is enabled.

11. All cloud inventory sources have the following update options:

- **Overwrite:** If checked, any hosts and groups that were previously present on the external source but are now removed, are removed from the automation controller inventory. Hosts and groups that were not managed by the inventory source are promoted to the next manually created group, or, if there is no manually created group to promote them into, they are left in the "all" default group for the inventory. When not checked, local child hosts and groups not found on the external source remain untouched by the inventory update process.
- **Overwrite variables:** If checked, all variables for child groups and hosts are removed and replaced by those found on the external source. When not checked, a merge is performed, combining local variables with those found on the external source.
- **Update on launch:** Each time a job runs using this inventory, refresh the inventory from the selected source before executing job tasks. To avoid job overflows if jobs are spawned faster than the inventory can synchronize, selecting this enables you to configure a **Cache Timeout** to previous cache inventory synchronizations for a certain number of seconds.

The **Update on launch** setting refers to a dependency system for projects and inventory, and does not specifically exclude two jobs from running at the same time.

If a cache timeout is specified, then the dependencies for the second job are created, and it uses the project and inventory update that the first job spawned.

Both jobs then wait for that project or inventory update to finish before proceeding. If they are different job templates, they can then both start and run at the same time, if the system has the capacity to do so. If you intend to use automation controller's provisioning callback feature with a dynamic inventory source, **Update on launch** must be set for the inventory group.

If you synchronize an inventory source that uses a project that has **Update On launch** set, then the project might automatically update (according to cache timeout rules) before the inventory update starts.

You can create a job template that uses an inventory that sources from the same project that the template uses. In such a case, the project updates and then the inventory updates (if updates are not already in progress, or if the cache timeout has not already expired).

12. Review your entries and selections. This enables you to configure additional details, such as schedules and notifications.

13. To configure schedules associated with this inventory source, click the **Schedules** tab:

- If schedules are already set up, then review, edit, enable or disable your schedule preferences.
- If schedules have not been set up, you can add a schedule.

Related information

[Define, create, and build execution environments](#)

[Add a new schedule](#)

Source from a project's source control management

An inventory that is sourced from a project means that it uses the SCM type from the project it is tied to. For example, if the project's source is from GitHub, then the inventory uses the same source.

Use the following procedure to configure a project-sourced inventory:

Procedure

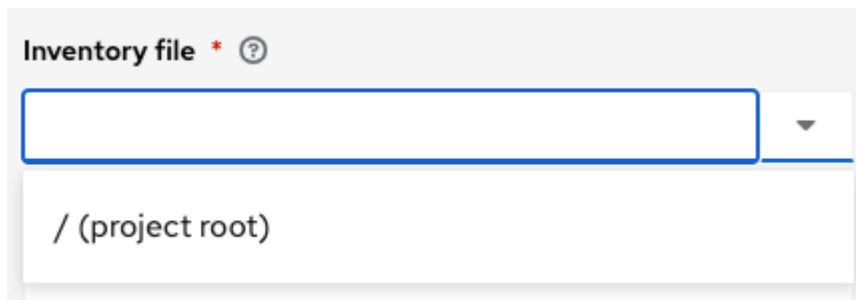
1. From the navigation panel, select **Automation Execution > Infrastructure > Inventories**.
2. Select the inventory name you want a source to and click the **Sources** tab.
3. Click **Create source**.
4. In the **Create source** page, select **Sourced from a Project** from the **Source** list.
5. Enter the following details in the additional fields:
 - Optional: **Source control branch/tag/commit**: Enter the SCM branch, tags, commit hashes, arbitrary refs, or revision number (if applicable) from the source control (Git or Subversion) to checkout.
This field only displays if the sourced project has the **Allow branch override** option checked. For further information, see [SSCM Types - Configuring playbooks to use Git and Subversion](#).

Options

☐ Clean ⓘ ☐ Delete ⓘ ☐ Track submodules ⓘ ☐ Update Revision on Launch ⓘ ☒ Allow Branch Override ⓘ

Some commit hashes and refs might not be available unless you also give a custom refspec in the next field. If left blank, the default is HEAD which is the last checked out Branch/Tag/Commit for this project.

- Optional: **Credential**: Specify the credential to use for this source.
- **Project** (required): Pre-populates with a default project, otherwise, specify the project this inventory is using as its source. Click the **Q** icon to choose from a list of projects. If the list is extensive, use the search to narrow the options.
- **Inventory file** (required): Select an inventory file associated with the sourced project. If not already populated, you can type it into the text field within the menu to filter extraneous file types. In addition to a flat file inventory, you can point to a directory or an inventory script.



Inventory file * ?

/ (project root)

6. Optional: You can specify the verbosity, host filter, enabled variable/value, and update options as described in [Adding a source](#).
7. Optional: To pass to the custom inventory script, you can set environment variables in the **Source variables** field. You can also place inventory scripts in source control and then run it from a project. For more information, see [Inventory File Importing](#).

If you are executing a custom inventory script from SCM, ensure that you set the execution bit (`chmod +x`) for the script in your upstream source control.

If you do not, automation controller throws a [Error 13] `Permission denied` error on execution.

Source an inventory from Amazon Web Services EC2

Use the following procedure to configure an AWS EC2-sourced inventory.

Procedure

1. From the navigation panel, select **Automation Execution > Infrastructure > Inventories**.
2. Select the inventory name you want a source to and click the **Sources** tab.
3. Click **Create source**.
4. In the **Create source** page, select **Amazon EC2** from the **Source** list.
5. The **Create source** window expands with additional fields. Enter the following details:
 - Optional: **Credential**: Choose from an existing AWS credential. For more information, see [Managing user credentials](#).
If automation controller is running on an EC2 instance with an assigned IAM Role, the credential can be omitted, and the security credentials from the instance metadata are used instead. For more information about using IAM Roles, see [IAM roles for Amazon EC2_documentation_at_Amazon](#) documentation at Amazon.
6. Optional: You can specify the verbosity, host filter, enabled variables or values, and update options as described in [Adding a source](#).
7. Use the **Source variables** field to override variables used by the `aws_ec2` inventory plugin. Enter variables by using either JSON or YAML syntax. Use the radio button to toggle

between the two. For more information about these variables, see the [aws inventory plugin documentation](#).

If you only use `include_filters`, the AWS plugin always returns all the hosts. To use this correctly, the first condition on the `or` must be on `filters` and then build the rest of the `OR` conditions on a list of `include_filters`.

Source an inventory from Google Compute Engine

Learn how to configure a Google-sourced inventory:

Procedure

1. From the navigation panel, select **Automation Execution > Infrastructure > Inventories**.
2. Select the inventory name you want a source to and click the **Sources** tab.
3. Click **Create source**.
4. In the **Add new source** page, select **Google Compute Engine** from the **Source** list.
5. The **Create source** window expands with the required **Credential** field. Choose from an existing GCE Credential. For more information, see [Managing user credentials](#).
6. Optional: You can specify the verbosity, host filter, enabled variables or values, and update options as described in [Adding a source](#).
7. Use the **Source Variables** field to override variables used by the `gcp_compute` inventory plugin. Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two. For more information about these variables, see the [gcp_compute inventory plugin documentation](#).

Source an inventory from Microsoft Azure resource manager

Use the following procedure to configure an Microsoft Azure Resource Manager-sourced inventory.

Procedure

1. From the navigation panel, select **Automation Execution > Infrastructure > Inventories**.
2. Select the inventory name you want a source to and click the **Sources** tab.
3. Click **Create source**.

4. In the **Create source** page, select **Microsoft Azure Resource Manager** from the **Source** list.
5. Enter the following details in the additional fields:
6. Optional: **Credential**: Choose from an existing Azure Credential. For more information, see [Managing user credentials](#).
7. Optional: You can specify the verbosity, host filter, enabled variables or values, and update options as described in [Adding a source](#).
8. Use the **Source variables** field to override variables used by the `azure_rm` inventory plugin. Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two. For more information about these variables, see the [azure_rm inventory plugin documentation](#).

Source an inventory from VMware vCenter

You can configure automation controller to synchronize inventory from a VMware vCenter server. You can manage virtual machines as part of your automation workflows.

Procedure

1. From the navigation panel, select **Automation Execution > Infrastructure > Inventories**.
2. Select the inventory name you want a source to and click the **Sources** tab.
3. Click **Create source**.
4. In the **Create source** page, select **VMware vCenter** from the **Source** list.
5. The **Create source** window expands with additional fields. Enter the following details:
 - Optional: **Credential**: Choose from an existing VMware credential. For more information, see [Managing user credentials](#).
6. Optional: You can specify the verbosity, host filter, enabled variables or values, and update options as described in [Adding a source](#).
7. Use the **Source Variables** field to override variables used by the `vmware_inventory` inventory plugin. Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two. For more information about these variables, see the [vmware_inventory inventory plugin](#).

VMWare properties have changed from lower case to camel case. Automation controller provides aliases for the top-level keys, but lower case keys in nested properties have been discontinued.

Source an inventory from VMware ESXi

Learn how to configure a VMWare ESXi sourced inventory.

Procedure

1. From the navigation panel, select **Automation Execution > Infrastructure > Inventories**.
2. Select the inventory name you want to add a source to, and click the **Sources** tab.
3. Click **Create source**.
4. Enter a **Name** for the source (required).
5. In the **Create source** page, select **VMware ESXi** from the **Source** list.
6. The **Create source** window expands with additional fields. Enter the following details:
 - **Credential:** Choose from an existing VMware credential. For more information, see [Managing user credentials](#).
7. Use the **Verbosity** menu to select the level of output on any inventory source's update jobs.
8. Optional: You can specify the host filter, enabled variables or values, and update options as described in [Adding a source](#).
9. Use the **Source Variables** field to override variables used by the `vmware_inventory` inventory plugin. Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two.

VMWare properties have changed from lower case to camel case. Automation controller provides aliases for the top-level keys, but lower case keys in nested properties have been discontinued.

Related information

[VMware ESXi plugin](#)

Source an inventory from Red Hat Satellite 6

automation controller can integrate with Red Hat Satellite 6 as a dynamic inventory source.

Use the following procedure to configure a Red Hat Satellite-sourced inventory.

Procedure

1. From the navigation panel, select **Automation Execution > Infrastructure > Inventories**.
2. Select the inventory name you want a source to and click the **Sources** tab.
3. Click **Create source**.

4. In the **Create source** page,, select **Red Hat Satellite 6** from the **Source** list.
5. The **Create source** window expands with additional fields. Enter the following details:
 - Optional: **Credential**: Choose from an existing Satellite Credential. For more information, see [Managing user credentials](#).
6. Optional: You can specify the verbosity, host filter, enabled variables or values, and update options as described in [Adding a source](#).
7. Use the **Source Variables** field to specify parameters used by the `foreman` inventory source. Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two.

If you meet an issue with the automation controller inventory not having the "related groups" from Satellite, you might need to define these variables in the inventory source. For more information, see [Red Hat Satellite 6](#).

If you see the message, `"no foreman.id" variable(s) when syncing the inventory`, see the solution on the Red Hat Customer Portal at: <https://access.redhat.com/solutions/5826451>. Be sure to login with your customer credentials to access the full article.

Source an inventory from Red Hat Lightspeed

You can create an inventory source that uses Red Hat Lightspeed as the source of hosts.

Use the following procedure to configure a Red Hat Lightspeed-sourced inventory.

Procedure

1. From the navigation panel, select **Automation Execution > Infrastructure > Inventories**.
2. Select the inventory name you want a source to and click the **Sources** tab.
3. Click **Create source**.
4. In the **Create source** page, select **Red Hat Lightspeed** from the **Source** list.
5. The **Create source** window expands with additional fields. Enter the following details:
 - Optional: **Credential**: Choose from an existing Red Hat Lightspeed Credential. For more information, see [Managing user credentials](#).
6. Optional: You can specify the verbosity, host filter, enabled variables or values, and update options as described in [Adding a source](#).
7. Use the **Source Variables** field to override variables used by the `Red Hat Lightspeed` inventory plugin. Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two. For more information about these variables, see [Red Hat Lightspeed inventory plugin](#).

Source an inventory from OpenStack

You can create an inventory source that uses the OpenStack inventory plugin to dynamically generate inventory from your OpenStack cloud.

Procedure

1. From the navigation panel, select **Automation Execution > Infrastructure > Inventories**.
2. Select the inventory name you want a source to and click the **Sources** tab.
3. Click **Create source**.
4. In the **Create source** page, select **OpenStack** from the **Source** list.
5. The **Create source** window expands with additional fields. Enter the following details:
 - Optional: **Credential**: Choose from an existing OpenStack Credential. For more information, see [Managing user credentials](#).
6. Optional: You can specify the verbosity, host filter, enabled variables or values, and update options as described in [Adding a source](#).
7. Use the **Source Variables** field to override variables used by the `openstack` inventory plugin. Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two.

Source an inventory from Red Hat Virtualization

Learn how to configure a Red Hat virtualization-sourced inventory.

Procedure

1. From the navigation panel, select **Automation Execution > Infrastructure > Inventories**.
2. Select the inventory name you want a source to and click the **Sources** tab.
3. Click **Create source**.
4. In the **Create source** page, select **Red Hat Virtualization** from the **Source** list.
5. The **Create source** window expands with additional fields. Enter the following details:
 - Optional: **Credential**: Choose from an existing Red Hat Virtualization Credential. For more information, see [Managing user credentials](#).
6. Optional: You can specify the verbosity, host filter, enabled variables or values, and update options as described in [Adding a source](#).

7. Use the **Source Variables** field to override variables used by the `ovirt` inventory plugin. Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two. For more information about these variables, see [ovirt inventory plugin](#)

NOTE:

Red Hat Virtualization (ovirt) inventory source requests are secure by default. To change this default setting, set the key `ovirt_insecure` to **true** in `source_variables`, which is only available from the API details of the inventory source at the `/api/v2/inventory_sources/N/` endpoint.

Source an inventory from Red Hat Ansible Automation Platform

An inventory that is sourced from Red Hat Ansible Automation Platform uses the Red Hat Ansible Automation Platform inventory plugin to gather inventory data from the Red Hat Ansible Automation Platform platform.

Use the following procedure to configure an automation controller-sourced inventory.

Procedure

1. From the navigation panel, select **Automation Execution > Infrastructure > Inventories**.
2. Select the inventory name you want a source to and click the **Sources** tab.
3. Click **Create source**.
4. In the **Create source** page, select **Red Hat Ansible Automation Platform** from the **Source** list.
5. The **Create source** window expands with additional fields. Enter the following details:
 - Optional: **Credential**: Choose from an existing Red Hat Ansible Automation Platform Credential. For more information, see [Managing user credentials](#).
6. Optional: You can specify the verbosity, host filter, enabled variables or values, and update options as described in [Adding a source](#).
7. Use the **Source Variables** field to override variables used by the `controller` inventory plugin. Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two. For more information about these variables, see [Controller inventory plugin](#). This requires your Red Hat Customer login.

Source an inventory from Terraform State

Use the following procedure to create a Terraform State inventory source.

This inventory source uses the `terraform_state` inventory plugin from the [cloud.terraform](#) collection. The plugin parses a terraform state file and add hosts for AWS EC2, GCE, and Microsoft Azure instances.

Procedure

1. From the navigation panel, select **Automation Execution > Projects**.
2. On the **Projects** page, click **Create project** to start the **Create project** window.
 - Enter the appropriate details according to the steps in [Adding a new project](#).
3. From the navigational panel, select **Automation Execution > Infrastructure > Inventories**.
4. Select the inventory that you want to add a source to.
5. In the **Sources** tab, click **Create source**.
6. From the **Source** menu, select **Terraform State**.
 - The **Create source** window expands with the optional **Credential** field. Choose an existing Terraform backend configuration credential. For more information, see [Terraform backend configuration](#).
7. Enable the options to **Overwrite** and **Update on launch**.
8. Use the **Source variables** field to override variables used by the `terraform_state` inventory plugin. Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two. For more information about these variables, see the [terraform_state](#) file.

The `backend_type` variable is required by the Terraform State inventory plugin. This should match the remote backend configured in the **Terraform backend credential**. The following is an example Amazon S3 backend:

```
backend_type: s3
```

9. Select an **Execution environment** that has a Terraform binary. This is required for the inventory plugin to run the Terraform commands that read inventory data from the Terraform state file.

IMPORTANT:

Terraform provider for Ansible Automation Platform inventories are managed by Terraform and you must not edit them in Ansible Automation Platform as it can introduce drift to the Terraform deployment.

Related information

[Terraform EE](#)

[Red Hat Ansible Automation Platform provider](#)

Source an inventory from OpenShift Virtualization

Learn how to add an OpenShift Virtualization inventory source to an existing inventory.

Before you begin

- You need a virtual machine deployed in a specific namespace and an OpenShift or Kubernetes API Bearer Token credential.

This inventory source uses a cluster that is able to deploy Red Hat OpenShift Container Platform Virtualization.

Procedure

1. From the navigational panel, select **Automation Execution > Infrastructure > Inventories**.
2. Select the inventory that you want to add a source to.
3. In the **Sources** tab, click **Add source**.
4. From the **Source** menu, select **OpenShift Virtualization**.
 - The **Add new source** window expands with the required **Credential** field. Choose from an existing Kubernetes API Bearer Token credential. For more information, see [OpenShift or Kubernetes API Bearer Token credential type](#). In this example, the `cmv2.engineering.redhat.com` credential is used.
5. You can optionally specify the **Verbosity**, **Host Filter**, **Enabled Variable/Value**, and **Update options** as described in the [Adding a source](#) steps.
6. Use the **Source Variables** field to override variables used by the `kubernetes` inventory plugin. Enter variables by using either JSON or YAML syntax. Use the radio button to toggle between the two. For more information about these variables, see the [kubevirt.core.kubevirt inventory source](#) documentation.

In the following example, the `connections` variable is used to specify access to a particular namespace in a cluster:

```
---
connections:
- namespaces:
  - hao-test
```

7. Click **Save** and then click **Sync** to sync the inventory.

Source an inventory from a custom inventory plugin

This describes how to use the `servicenow.itsm` collection inventory plugin to sync inventory on Ansible Automation Platform.

Procedure

1. Create and sync a project using a Source Control Git repository that includes the following files:

```
>> requirements.yml
---
collections: - name: servicenow.itsm

>> inventories/myinventory.now.yml

# Create a file following the example below. It must have a configuration file
# extension ending in either "now.yml" or "now.yaml".

plugin: servicenow.itsm.now
query:
  - os: = Linux Red Hat
  - os: = Windows XP
keyed_groups:
  - key: os
  prefix: os
```

NOTE:

Refer to the official [Ansible documentation](#) for detailed guidance on using and configuring the `servicenow.itsm.now` plugin.

2. Create an inventory by setting the source to **Sourced from a Project**, selecting the new project, and choose `/(project root)` in the inventory file section.
3. Synchronize the source in the inventory.

Export old inventory scripts

Despite the removal of the custom inventory scripts API, the scripts are still saved in the database.

To recover the scripts from the database in a format that is suitable for you to check into source control, use the following command:

```
$ awx-manage export_custom_scripts --filename=my_scripts.tar
```

Dump of old custom inventory scripts at my_scripts.tar

Making use of the output:

```
$ mkdir my_scripts
```

```
$ tar -xf my_scripts.tar -C my_scripts
```

The name of the scripts has the form: `<pk>_<name>`. This is the naming scheme used for project folders.

```
$ ls my_scripts
```

```

_10inventory_script_rawhook                _19
_30inventory_script_listenhospital
_11inventory_script_upperorder              _1inventory_script_commercialinternet45  _4i
nventory_script_whitestring
_12inventory_script_eastplant               _22inventory_script_pinexchange          _5i
nventory_script_literaturepossession
_13inventory_script_governmentculture       _23inventory_script_brainluck            _6i
nventory_script_opportunitytelephone
_14inventory_script_bottomguess             _25inventory_script_buyerleague         _7i
nventory_script_letjury
_15inventory_script_wallisland              _26inventory_script_lifesport           _8r
andom_inventory_script
_16inventory_script_wallisland              _27inventory_script_exchangesomewhere   _9r
andom_inventory_script
_17inventory_script_bidstory                _28inventory_script_boxchild
_18p                                         _29__inventory_script_wearstress

```

Each file contains a script. Scripts can be `bash/python/ruby/more`, so the extension is not included. They are all directly executable. Executing the script dumps the inventory data.

```
$ ./my_scripts/11__inventory_script_upperorder

{"group":
"\ud801\udcb0\uc20e\u7b0e\ud81c\udfeb\ub12b\ub4d0\u9ac6\ud81e\udf07\u6ff9\uc17b":
{"hosts":

["host_\ud821\udcad\u68b6\u7a51\u93b4\u69cf\uc3c2\ud81f\uddbe\ud820\udc92\u3143\u62c7",
"host_\u6057\u3985\u1f60\ufefb\u1b22\u2d2d\ua90c\ud81a\udc69\u1344\u9d15",
"host_\u78a0\ud820\udf3\u925e\u69da\ua549\ud80c\u2d7e\ud81e\udc91\ud808\u2d11\u57d6\ud801\u2d57",
"host_\ud83a\udc2d\ud7f7\ua18a\u779a\ud800\udf8b\u7903\ud820\udead\u4154\ud808\u2d15\u9711",
"host_\u18a1\u9d6f\u08ac\u74c2\u54e2\u740e\u5f02\ud81d\u2d2e\u2d46\u4506"], "vars":
{"ansible_host": "127.0.0.1", "ansible_connection":

"local"}}}
```

You can verify functionality with `ansible-inventory`. This gives the same data, but reformatted.

```
$ ansible-inventory -i ./my_scripts/_11__inventory_script_upperorder --list --export
```

In the preceding example, you can `cd` into `my_scripts` and then issue a `git init` command, add the scripts you want, push it to source control, and then create an SCM inventory source in the user interface.

For more information about syncing or using custom inventory scripts, see [See "Import your inventory file from source control" on page 156](#) in *Configuring automation execution*.

Run ad hoc commands against a host in an inventory

Learn how to run ad hoc commands against hosts in an inventory.

Ad hoc refers to using Ansible to perform a quick command, using `/usr/bin/ansible`, rather than the orchestration language, which is `/usr/bin/ansible-playbook`. An example of an ad hoc command might be rebooting 50 machines in your infrastructure. Anything you can do ad hoc can be accomplished by writing a playbook. Playbooks can also glue many other operations together.

Procedure

1. From the navigation panel, select **Automation Execution > Infrastructure > Inventories**.
2. Select the inventory name you want to run an ad hoc command with.
3. Select an inventory source from the **Hosts** or **Groups** tab. The inventory source can be a single group or host, a selection of many hosts, or a selection of many groups.
4. Click **Run Command**. The Run command window opens.

5. Enter the following information:

- **Module:** Select one of the modules that the supports running commands against.

command	apt_repository	mount	win_service
shell	apt_rpm	ping	win_updates
yum	service	SELinux	win_group
apt	group	setup	win_user
apt_key	user	win_ping	win_user

- **Arguments:** Provide arguments to be used with the module you selected.
 - **Limit:** Enter the limit used to target hosts in the inventory. To target all hosts in the inventory enter `all` or `*`, or leave the field blank. This is automatically populated with whatever was selected in the previous view before clicking the launch button.
 - **Machine Credential:** Select the credential to use when accessing the remote hosts to run the command. Choose the credential containing the username and SSH key or password that Ansible needs to log in to the remote hosts.
 - **Verbosity:** Select a verbosity level for the standard output.
 - **Forks:** If needed, select the number of parallel or simultaneous processes to use while executing the command.
 - **Show Changes:** Select to enable the display of Ansible changes in the standard output. The default is OFF.
 - **Enable Privilege Escalation:** If enabled, the playbook is run with administrator privileges. This is the equivalent of passing the `--become` option to the `ansible` command.
 - **Extra Variables:** Provide extra command line variables to be applied when running this inventory. Enter variables using either JSON or YAML syntax. Use the radio button to toggle between the two.
6. Click **Next** to select the execution environment you want the ad hoc command to be run against.
 7. Click **Next** to select the credential you want to use.
 8. Click **Launch**. The results display in the **Output** tab of the module's job window.

Configure notifications for the source

After you create a source for an inventory in automation controller, you can configure notifications for that source.

Procedure

1. From the navigation panel, select **Automation Execution > Infrastructure > Inventories**.
2. Select the inventory name you want to configure notifications for.
3. In the inventory **Details** page, select the **Notifications** tab.

NOTE:

The **Notifications** tab is only present when you have saved the newly-created source.

4. If notifications are already set up, use the toggles to enable or disable the notifications to use with your particular source. For more information, see [Enable and Disable Notifications](#).
5. If you have not set up notifications, see [Notifiers](#) for more information.
6. Review your entries and selections.
7. Click **Save**.

Next steps

When you define a source, it is displayed in the list of sources associated with the inventory. From the **Sources** tab you can perform a sync on a single source, or sync all of them at once. You can also perform additional actions such as scheduling a sync process, and edit or delete the source.

View completed jobs

You can view completed jobs in automation controller by navigating to the **Jobs** tab in the navigation panel.

If you use an inventory to run a job, you can view details about those jobs in the **Jobs** tab of the inventory and click **Expanded** to view details about each job.

Import your inventory file from source control

With automation controller you can select an inventory file from source control, rather than creating one from scratch.

The files are non-editable, and as inventories are updated at the source, the inventories within the projects are also updated accordingly, including the `group_vars` and `host_vars` files or directory associated with them. SCM types can consume both inventory files and scripts. Both inventory files and custom inventory types use scripts.

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

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

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

Related information

[Export old inventory scripts](#)

Inventory file fields for source control

You can use a source control management (SCM) system, such as Git, to manage your inventory files in automation controller. You can use an SCM system to version control your inventory files and manage them in a collaborative manner.

The source fields used are:

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

Additionally:

- An update of the project automatically triggers an inventory update where it is used.

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

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

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

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

The status indicator links to the project update job.

The project name links to the project.

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

Supported file syntax

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

IMPORTANT:

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

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

```
{
  "_meta": {
    "hostvars": {
      "host1": {
        "fly_rod": true
      }
    }
  },
  "all": {
    "children": [
      "groupA",
```

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

Related information

[test-playbooks/inventories](#)
[inventories/changes.py](#)

[How to migrate inventory scripts from Red Hat Ansible tower to Red Hat Ansible Automation Platform?](#)

Define where automation runs with host and node groupings

Managing hosts and controlling where automation jobs run ensures efficient resource utilization and proper workload distribution across your infrastructure. Define which systems to manage and organize execution capacity to align with your operational requirements.

Defining where automation runs helps you to:

- **Target the right systems:** Organize hosts into inventories to define which systems to manage and ensure automation runs against the correct infrastructure components.
- **Control where jobs run:** Group instances into instance groups and assign them to resources to control where jobs run and manage capacity.
- **Optimize resource allocation:** Balance automation workloads across available capacity by configuring how instance groups distribute jobs.

How Ansible Automation Platform supports host and node groupings

Ansible Automation Platform uses two grouping mechanisms:

- **Hosts and inventories** define which systems you want to manage. A host is a system managed by Ansible Automation Platform. It can be a physical server, a virtual machine, a cloud-based server, or another device. Group hosts in inventories and use patterns to select which hosts or groups to target when running automation.
- **Instance groups** organize instances together to control where automation jobs run. Assign instance groups to organizations, inventories, or job templates. When you launch a job, the platform checks for instance groups associated with the job template, then the inventory, then the organization in that order. The platform then runs the job on the instance group with available capacity.

Add new hosts as automation targets

A host is a system managed by Ansible Automation Platform, which might include a physical, virtual, cloud-based server, or other device.

Typically a host is an operating system instance.

Hosts are grouped in inventories and are sometimes referred to as a “nodes”.

Ansible works against multiple managed nodes or “hosts” in your infrastructure at the same time, using a list or group of lists known as an inventory.

Once your inventory is defined, use patterns to select the hosts or groups you want Ansible to run against.

Create a host

Learn how to create a new host by providing a name, selecting an inventory, and optionally defining a description and associated variables.

To create a new host.

Procedure

1. From the navigation panel, select **Automation Execution > Infrastructure > Hosts**.
2. Click **Create host**.
3. On the **Create Host** page enter the following information:
 - **Name**: Enter a name for your host.
 - (Optional) **Description**: Enter a description for your host.
 - **Inventory**: Select the inventory for this host to belong to.
 - **Variables**: Enter the inventory file variables associated with your host.
4. Click **Create host** to save your changes.

View the host details

Examine the Host details shown below for a job run.

Procedure

1. From the navigation panel, select **Automation Execution > Infrastructure > Hosts**. The **Hosts** page displays the following information about the host or hosts affected by recent job runs.
2. Selecting a particular host displays the **Details** page for that host, with the following information:
 - The **Name** of the Host.

- The **Inventory** associated with that host. Selecting this inventory displays details of the inventory.
- When the Host was **Created** and by whom. Selecting the creator displays details of the creator.
- When the Host was **Last modified**. Selecting the creator displays details of the creator.
- **Variables** associated with the Host. You can display the variables in YAML or JSON format.

3. Click **Edit host** to edit details of the host.

- Select the **Facts** tab to display facts associated with the host.
- Select the **Groups** tab to display the Groups associated with the host.
 - Click **Associate groups** to associate a group with the host.
- Select the **Jobs** tab to display the Jobs which ran on the host.
 - Click the ➤ icon to display details of the job.

Hosts > localhost > Jobs

localhost Host enabled [Edit host](#)

Back to Hosts Details Facts Groups **Jobs**

Sort Finish Time

ID	Name	Status	Start Time	Finish Time
1	Demo Job Template	Success	7/5/2024, 12:22:35 PM	7/5/2024, 12:22:48 PM

Launched by: dev
 Job template: Demo Job Template
 Source workflow job
 Inventory: Demo Inventory
 Project: Demo Project
 Execution environment: Default execution environment
 Job slice: 0/1

Determine where automation runs with instance groups

An Instance Group enables you to group instances in a clustered environment. Policies dictate how instance groups behave and how jobs are executed. The following view displays the capacity levels based on policy algorithms:

Instance Groups ⓘ

An instance group provides the ability to group instances in a clustered environment.

Create group

Sort Name

Name	Type	Running jobs	Total jobs	Instances	Used capacity
controlplane	Instance group	0	1	1	0%
default	Instance group	0	0	1	0%

Related information

[Instance groups](#)

[Container groups](#)

Instance groups

Instances can be grouped into one or more instance groups. Instance groups can be assigned to one or more of the following listed resources:

- Organizations
- Inventories
- Job templates

When a job associated with one of the resources executes, it is assigned to the instance group associated with the resource. During the execution process, instance groups associated with job templates are checked before those associated with inventories. Instance groups associated with inventories are checked before those associated with organizations. Therefore, instance group assignments for the three resources form the hierarchy:

Job Template > Inventory > Organization

Consider the following when working with instance groups:

- You can define other groups and group instances in those groups. These groups must be prefixed with `instance_group_`. Instances are required to be in the `automationcontroller` or `execution_nodes` group alongside other `instance_group_` groups. In a clustered setup, at least one instance must be present in the `automationcontroller` group, which is displayed as `controlplane` in the API instance groups.
- You cannot modify the `controlplane` instance group, and attempting to do so results in a permission denied error for any user. Therefore, the **Disassociate** option is not available in the **Instances** tab of `controlplane`.
- A `default` API instance group is automatically created with all nodes capable of running jobs. This is similar to any other instance group but if a specific instance group is not associated with a specific resource, then the job execution always falls back to the default instance group. The default instance group always exists, and you cannot delete or rename it.
- Do not create a group named `instance_group_default`.
- Do not name any instance the same as a group name.

Create an instance group

You can create instance groups with Automation controller to organize and manage your instances.

Use the following procedure to create a new instance group.

Procedure

1. From the navigation panel, select **Automation Execution > Infrastructure > Instance Groups**.
2. Click **Create group** and select **Create instance group** from the list.
3. Enter the appropriate details into the following fields:
 - **Name:** Names must be unique and must not be named "controller".
 - **Policy instance minimum:** Enter the minimum number of instances to automatically assign to this group when new instances come online.
 - **Policy instance percentage:** Use the slider to select a minimum percentage of instances to automatically assign to this group when new instances come online.

NOTE:

Policy instance fields are not required to create a new instance group. If you do not specify values, then the **Policy instance minimum** and **Policy instance percentage** default to 0.

- **Max concurrent jobs:** Specify the maximum number of forks that can be run for any given job.
- **Max forks:** Specify the maximum number of concurrent jobs that can be run for any given job.

NOTE:

The default value of 0 for **Max concurrent jobs** and **Max forks** denotes no limit. For more information, see [Instance group capacity limits](#).

4. Click **Create instance group**, or, if you have edited an existing Instance Group click **Save instance group**

Next steps

When you have successfully created the instance group the **Details** tab of the newly created instance group enables you to review and edit your instance group information.

You can also edit **Instances** and review **Jobs** associated with this instance group:

Instance groups > Test group > Details

Test group

Edit instance group

Back to Instance Groups

Details

Instances

Jobs

Name	Type	Policy Instance Minimum ⓘ
Test group	Instance Group	0
Policy instance percentage ⓘ	Max concurrent jobs ⓘ	Max forks ⓘ
0%	2	6
Used Capacity	Created	Last modified
0	5/22/2024, 10:25:01 AM	5/22/2024, 10:25:01 AM

Associate instances to an instance group

Learn how to associate instances to an instance group in automation controller.

Procedure

1. Select the **Instances** tab on the **Details** page of an Instance Group.
2. Click **Associate instance**.
3. Click the checkbox next to one or more available instances from the list to select the instances you want to associate with the instance group and click **Confirm**

View jobs associated with an instance group

You can view jobs associated with an instance group to monitor their status and details.

Procedure

1. Select the **Jobs** tab of the **Instance Group** window.
2. Click the arrow ➤ icon next to a job to expand the view and show details about each job.
Each job displays the following details:

- The job status
- The ID and name
- The type of job
- The time it started and completed

- Who started the job and applicable resources associated with it, such as the template, inventory, project, and execution environment

Related information

[Instance group policies](#)

Control where an automation job runs

Use instance groups to control job execution locations. While the controller automatically selects groups based on capacity, you can override this by associating groups with job templates, inventories, or organizations.

When a job is launched, automation controller checks for any instance groups associated with the job template, inventory, or organization (by way of project) in that order.

If you associate instance groups with a job template, inventory, or organization, a job run from that job template is not eligible for the default behavior. This means that if all of the instances inside of the instance groups associated with these three resources are out of capacity, the job remains in the pending state until capacity becomes available.

The order of preference in determining which instance group to submit the job to is as follows:

1. Job template
2. Inventory
3. Organization (by way of project)

If you associate instance groups with the job template, and all of these are at capacity, then the job is submitted to instance groups specified on the inventory, and then the organization. Jobs must run in those groups in preferential order as resources are available.

You can still associate the global `default` group with a resource, such as any of the custom instance groups defined in the playbook. You can use this to specify a preferred instance group on the job template or inventory, but still enable the job to be submitted to any instance if those are out of capacity.

- If you associate `group_a` with a job template and also associate the `default` group with its inventory, you enable the `default` group to be used as a fallback in case `group_a` gets out of capacity.
- In addition, it is possible to not associate an instance group with one resource but choose another resource as the fallback. For example, not associating an instance group with a job template and having it fall back to the inventory or the organization's instance group.

This presents the following possibilities:

1. Associating instance groups with an inventory (omitting assigning the job template to an instance group) ensures that any playbook run against a specific inventory runs only on the

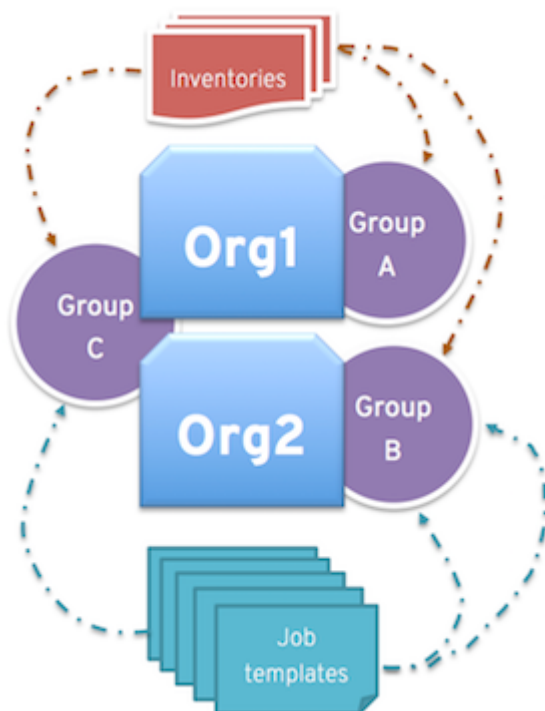
group associated with it. This is useful in the situation where only those instances have a direct link to the managed nodes.

2. An administrator can assign instance groups to organizations.

This enables the administrator to segment out the entire infrastructure and guarantee that each organization has capacity to run jobs without interfering with any other organization's ability to run jobs.

An administrator can assign multiple groups to each organization, similar to the following scenario:

- There are three instance groups: *A*, *B*, and *C*. There are two organizations: *Org1* and *Org2*.
- The administrator assigns group *A* to *Org1*, group *B* to *Org2* and then assigns group *C* to both *Org1* and *Org2* as an overflow for any extra capacity that might be needed.
- The organization administrators are then free to assign inventory or job templates to whichever group they want, or let them inherit the default order from the organization.



Arranging resources this way offers you flexibility. You can also create instance groups with only one instance, enabling you to direct work towards a very specific Host in the automation controller cluster.

Define instance groups in the inventory file

Define custom instance groups in the inventory file to organize nodes in your automation mesh. By correctly assigning nodes to the `automationcontroller` or `execution_nodes` groups, you ensure the installer validates the mesh topology and prevents configuration errors.

Use the following criteria when defining nodes:

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

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

The current behavior expects a member of an `instance_group_*` to be part of `automationcontroller` or `execution_nodes` group.

Define instance groups

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

[automationcontroller:vars]
peers=execution_nodes

[execution_nodes]

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

After you run the installation program, the following error is displayed:

```
TASK [ansible.automation_platform_installer.check_config_static : Validate mesh
topology] ***

fatal: [126-addr.tatu.home -> localhost]: FAILED! => {"msg": "The host '110-
addr.tatu.home' is not present in either [automationcontroller] or
[execution_nodes]"}
```

To fix this, move the box `110-addr.tatu.home` to an `execution_node` group, as follows:

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

[automationcontroller:vars]
peers=execution_nodes

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

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

This results in:

```
TASK [ansible.automation_platform_installer.check_config_static : Validate mesh
topology] ***

ok: [126-addr.tatu.home -> localhost] => {"changed": false, "mesh": {"110-
addr.tatu.home": {"node_type": "execution", "peers": [],
"receptor_control_filename": "receptor.sock", "receptor_control_service_name":
"control", "receptor_listener": true, "receptor_listener_port": 8928,
"receptor_listener_protocol": "tcp", "receptor_log_level": "info"}, "126-
addr.tatu.home": {"node_type": "control", "peers": ["110-addr.tatu.home"],
"receptor_control_filename": "receptor.sock", "receptor_control_service_name":
"control", "receptor_listener": false, "receptor_listener_port": 27199,
"receptor_listener_protocol": "tcp", "receptor_log_level": "info"}}
```

After upgrading from automation controller 4.0 or earlier, the legacy `instance_group_` member likely has the awx code installed. This places that node in the `automationcontroller` group.

Configure instance groups from the API

You can configure instance groups in automation controller using the REST API.

You can create instance groups by POSTing to `/api/v2/instance_groups` as a system administrator.

Once created, you can associate instances with an instance group by using:

```
HTTP POST /api/v2/instance_groups/x/instances/ {'id': y}`
```

An instance that is added to an instance group automatically reconfigures itself to listen on the group's work queue. For more information, see *Instance group policies*.

Related information

[Instance group policies](#)

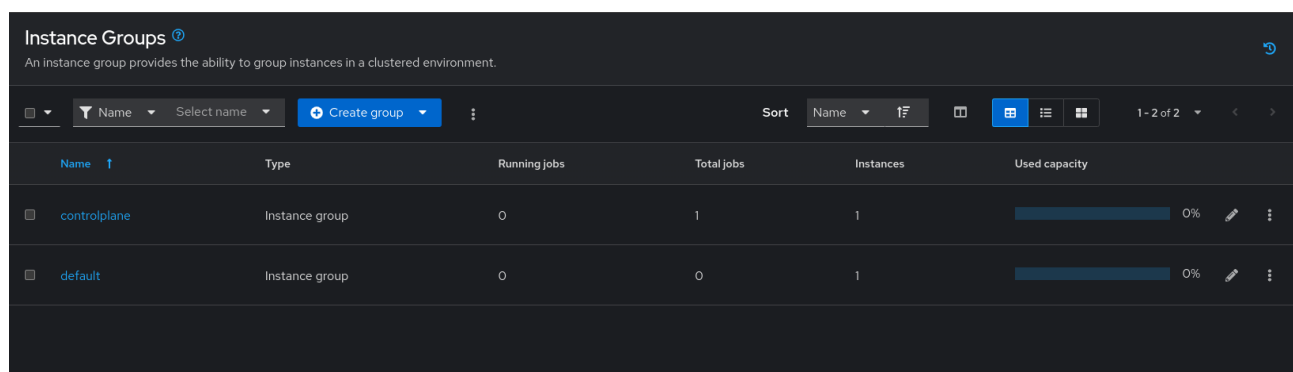
Instance group policies

You can configure automation controller instances to automatically join instance groups when they come online by defining a policy. These policies are evaluated for every new instance that comes online.

Instance group policies are controlled by the following three optional fields on an `Instance Group` :

- `policy_instance_percentage` : This is a number between 0 - 100. It guarantees that this percentage of active automation controller instances are added to this instance group. As new instances come online, if the number of instances in this group relative to the total number of instances is less than the given percentage, then new ones are added until the percentage condition is satisfied.
- `policy_instance_minimum` : This policy attempts to keep at least this many instances in the instance group. If the number of available instances is lower than this minimum, then all instances are placed in this instance group.
- `policy_instance_list` : This is a fixed list of instance names to always include in this instance group.

The **Instance Groups** list view from the automation controller user interface (UI) provides a summary of the capacity levels for each instance group according to instance group policies:



Name	Type	Running jobs	Total jobs	Instances	Used capacity
controlplane	Instance group	0	1	1	0%
default	Instance group	0	0	1	0%

Related information

[Managing Instance Groups](#)

Job slice considerations

When setting up job slices, consider the following:

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

[Jobs](#) > [Demo Job Template](#) > [Details](#)

[Relaunch job](#) [Cancel job](#) ⋮

Demo Job Template

◀ Back to Jobs Output Details

ID	Name	Status
7	Demo Job Template	✓ Success
Type	Labels ⓘ	Duration
Playbook run		3s
Started	Finished	Launched by
5/27/2025, 3:06:22 PM	5/27/2025, 3:06:25 PM	admin
Job template	Inventory ⓘ	Project ⓘ
Demo Job Template	Demo Inventory	Demo Project
Execution environment ⓘ	Credentials ⓘ	Job slice ⓘ
Default execution environment	SSH: Demo Credential	0/1

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

NOTE:

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

Normally, the number of slices is equal to or less than the number of hybrid or execution nodes. A job template can also be limited to certain instances through its instance groups. You can set an extremely high number of job slices but it can cause performance degradation. The job scheduler is not designed to simultaneously schedule thousands of workflow nodes, which are what the sliced jobs become.

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

- Any job that intends to orchestrate across hosts (rather than just applying changes to individual hosts) must not be configured as a slice job.

- If a sliced job fails, automation controller does not attempt to discover or account for the specific failed playbooks.

Job runtime behavior

When you run a job associated with an instance group, note the following behaviors:

- If you divide a cluster into separate instance groups, then the behavior is similar to the cluster as a whole.
- If you assign two instances to a group then either one is as likely to receive a job as any other in the same group.
- As automation controller instances are brought online, it effectively expands the work capacity of the system.
- If you place those instances into instance groups, then they also expand that group's capacity.
- If an instance is performing work and it is a member of multiple groups, then capacity is reduced from all groups for which it is a member.
- De-provisioning an instance removes capacity from the cluster wherever that instance was assigned. For more information, see the [Deprovisioning instance groups](#) section.

NOTE:

Not all instances are required to be provisioned with an equal capacity.

Pin instances manually to specific instance groups

If you have a special instance that needs to be only assigned to a specific instance group but do not want it to automatically join other groups by "percentage" or "minimum" policies, you can pin an instance manually to specific groups.,

Procedure

1. Add the instance to one or more instance groups' `policy_instance_list`.
2. Update the instance's `managed_by_policy` property to be `False`.

This prevents the instance from being automatically added to other groups based on percentage and minimum policy. It only belongs to the groups you have manually assigned it to:

```

HTTP PATCH /api/v2/instance_groups/N/
{
  "policy_instance_list": ["special-instance"]
}
HTTP PATCH /api/v2/instances/X/
{
  "managed_by_policy": False
}

```

Capacity limits for instance groups

There is external business logic that can drive the need to limit the concurrency of jobs sent to an instance group, or the maximum number of forks to be consumed.

For traditional instances and instance groups, you might want to enable two organizations to run jobs on the same underlying instances, but limit each organization's total number of concurrent jobs. This can be achieved by creating an instance group for each organization and assigning the value for `max_concurrent_jobs`.

For automation controller groups, automation controller is generally not aware of the resource limits of the OpenShift cluster. You can set limits on the number of pods on a namespace, or only resources available to schedule a certain number of pods at a time if no auto-scaling is in place. In this case, you can adjust the value for `max_concurrent_jobs`.

Another parameter available is `max_forks`. This provides additional flexibility for capping the capacity consumed on an instance group or container group. You can use this if jobs with a wide variety of inventory sizes and "forks" values are being run. You can limit an organization to run up to 10 jobs concurrently, but consume no more than 50 forks at a time:

```

max_concurrent_jobs: 10
max_forks: 50

```

If 10 jobs that use 5 forks each are run, an eleventh job waits until one of these finishes to run on that group (or be scheduled on a different group with capacity).

If 2 jobs are running with 20 forks each, then a third job with a `task_impact` of 11 or more waits until one of these finishes to run on that group (or be scheduled on a different group with capacity).

For container groups, using the `max_forks` value is useful given that all jobs are submitted using the same `pod_spec` with the same resource requests, irrespective of the "forks" value of the job. The default `pod_spec` sets requests and not limits, so the pods can "burst" above their requested value without being throttled or reaped. By setting the `max_forks` value, you can help prevent a

scenario where too many jobs with large forks values get scheduled concurrently and cause the OpenShift nodes to be oversubscribed with multiple pods by using more resources than their requested value.

To set the maximum values for the concurrent jobs and forks in an instance group, see [Creating an instance group](#).

Deprovision instance groups

Re-running the setup playbook won't deprovision instances, as clusters can't distinguish between intentional offline status or failure. Instead, shut down all services on the automation controller instance, then run the deprovisioning tool from another instance.

Procedure

1. Shut down the instance or stop the service with the following command:

```
automation-controller-service stop
```

2. Run the following deprovision command from another instance to remove it from the controller cluster registry:

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

For example:

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

Next steps

Deprovisioning instance groups in automation controller does not automatically deprovision or remove instance groups, even though re-provisioning often causes these to be unused. They can still show up in API endpoints and stats monitoring. You can remove these groups with the following command:

```
awx-manage unregister_queue --queueName=<name>
```

Removing an instance's membership from an instance group in the inventory file and re-running the setup playbook does not ensure that the instance is not added back to a group. To be sure that an instance is not added back to a group, remove it through the API and also remove it in your inventory file. You can also stop defining instance groups in the inventory file. You can manage

instance group topology through the automation controller UI. For more information about managing instance groups in the UI, see [Managing Instance Groups](#).

NOTE:

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

Control where automation runs with container groups

Ansible Automation Platform supports container groups, which enable you to run jobs in automation controller regardless of whether automation controller is installed as a standalone, in a virtual environment, or in a container.

Container groups act as a pool of resources within a virtual environment.

You can create instance groups to point to an OpenShift container. These are job environments that are provisioned on-demand as a pod that exists only for the duration of the playbook run. This is known as the ephemeral execution model and ensures a clean environment for every job run.

In some cases, you might want to set container groups to be "always-on", which you can configure through the creation of an instance.

NOTE:

Container groups upgraded from versions before automation controller 4.0 revert back to default and remove the old pod definition, clearing out all custom pod definitions in the migration.

Container groups are different from execution environments in that execution environments are container images and do not use a virtual environment. For more information, see *Define, create, and build execution environments* in the Related Links section.

Related information

[Define, create, and build execution environments](#)

Create a container group

You can create a `ContainerGroup` in automation controller to run jobs in containers on an OpenShift or Kubernetes cluster.

Before you begin

- A namespace that you can launch into. Every cluster has a "default" namespace, but you can use a specific namespace.
- A service account that has the roles that enable it to launch and manage pods in this namespace. For more information, see [Creating a service account in OpenShift Container Platform or Kubernetes](#).
- If you are using execution environments in a private registry, and have a container registry credential associated with them in automation controller, the service account also needs the roles to get, create, and delete secrets in the namespace. If you do not want to give these roles to the service account, you can pre-create the `ImagePullSecrets` and specify them on the pod spec for the `ContainerGroup`. In this case, the execution environment must not have a container registry credential associated, or automation controller attempts to create the secret for you in the namespace.
- A token associated with that service account. An OpenShift or Kubernetes Bearer Token.
- A CA certificate associated with the cluster.

A `ContainerGroup` is a type of `InstanceGroup` that has an associated credential you can use to connect to an OpenShift cluster.

Procedure

1. From the navigation panel, select **Automation Execution > Infrastructure > Instance Groups**.
2. Click **Create group** and select **Create container group**.
3. Enter a name for your new container group and select the credential you created before to associate it to the container group.
4. Click **Create container group**.
5. Check the **Customize pod spec** box and edit the **Pod spec override** to include the namespace and service account name that you used in the previous steps.

Create a service account in OpenShift Container Platform or Kubernetes

Use service accounts in an OpenShift cluster or Kubernetes to run jobs in a container group through automation controller. After the service account is created, its credentials are provided to automation controller in the form of an OpenShift or Kubernetes API Bearer Token credential.

Procedure

1. To create a service account, download and use the following sample service account example, `containergroup sa` and change it as required to obtain the credentials:

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: containergroup-service-account
  namespace: containergroup-namespace
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: role-containergroup-service-account
  namespace: containergroup-namespace
rules:
  - verbs:
      - get
      - list
      - watch
      - create
      - update
      - patch
      - delete
    apiGroups:
      - ''
    resources:
      - pods
```

```

- verbs:
  - get
  apiGroups:
    - ''
  resources:
    - pods/log
- verbs:
  - create
  apiGroups:
    - ''
  resources:
    - pods/attach
- verbs:
  - get
  - create
  - delete
  apiGroups:
    - ''
  resources:
    - secrets
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: role-containergroup-service-account-binding
  namespace: containergroup-namespace
subjects:
- kind: ServiceAccount
  name: containergroup-service-account
  namespace: containergroup-namespace
roleRef:
  kind: Role
  name: role-containergroup-service-account
  apiGroup: rbac.authorization.k8s.io

```

2. Apply the configuration from `containergroup-sa.yml`:

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

3. Get an API token by generating a service account token:

```
oc create token containergroup-service-account --duration=$((365*24))h > containergroup-sa.token
```

4. Get the CA certificate:

```
oc get secret -n openshift-ingress wildcard-tls -o jsonpath='{.data.ca\.crt}' | base64 -d > containergroup-ca.crt
```

5. Use the contents of `containergroup-sa.token` and `containergroup-ca.crt` to provide the information for the [OpenShift or Kubernetes API Bearer Token](#) required for the container group.
 - a. To create a container group, create a [OpenShift or Kubernetes API Bearer Token](#) credential to use with your container group.

Related information

[Creating new credentials](#)

Customize the pod specification

Ansible Automation Platform provides a simple default pod specification, however, you can provide a custom YAML or JSON document that overrides the default pod specification.

This field uses any custom fields such as `ImagePullSecrets`, that can be "serialized" as valid pod JSON or YAML. A full list of options can be found in the [Pods and Services](#) section of the OpenShift documentation.

Procedure

1. From the navigation panel, select **Automation Execution > Infrastructure > Instance Groups**.
2. Click **Create group** and select **Create container group**.
3. Check the option for **Customize pod spec**.
4. Enter a custom Kubernetes or OpenShift Pod specification in the **Pod spec override** field.

Instance Groups > Create Container Group

Create Container Group

Name * **Credential** **Max concurrent jobs**
Maximum number of jobs to run concurrently on this group. Zero means no limit will be enforced.

Max forks
Maximum number of forks to allow across all jobs running concurrently on this group. Zero means no limit will be enforced.

☒ Customize pod spec

Pod spec override YAML JSON

5. Click **Create container group**.

NOTE:

The image when a job launches is determined by which execution environment is associated with the job. If you associate a container registry credential with the execution environment, then automation controller attempts to make an `ImagePullSecret` to pull the image. If you prefer not to give the service account permission to manage secrets, you must pre-create the `ImagePullSecret` and specify it on the pod specification, and omit any credential from the execution environment used.

For more information, see the [Allowing Pods to Reference Images from Other Secured Registries](#) section of the *Red Hat Container Registry Authentication* article.

6. When you have created the container group successfully, the **Details** tab of the newly created container group remains, which enables you to review and edit your container group information. This is the same menu that is opened if you click the icon from the **Instance Groups** list view.

You can also edit **Instances** and review **Jobs** associated with this instance group.

Instance groups > default > Details

default Edit instance group

[Back to Instance Groups](#) **Details** [Instances](#) [Team access](#) [User access](#) [Jobs](#)

Name	Type	Policy instance minimum ⓘ
default	Instance Group	0
Policy instance percentage ⓘ	Max concurrent jobs ⓘ	Max forks ⓘ
0%	0	0
Used Capacity	Created	Last modified
0	5/17/2024, 6:18:32 AM	5/17/2024, 6:18:32 AM

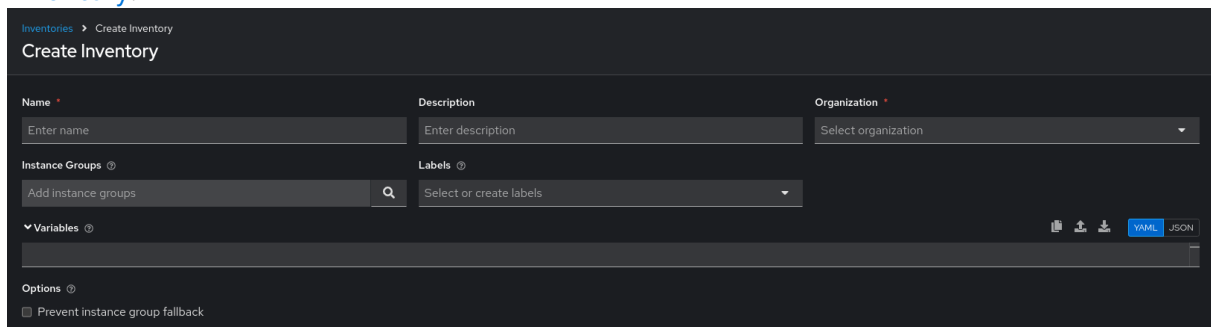
Container groups and instance groups are labeled accordingly.

Verify the container group functions

Verify the deployment and termination of your container.

Procedure

1. Create a mock inventory and associate the container group to it by populating the name of the container group in the **Instance groups** field. For more information, see [Add a new inventory](#).

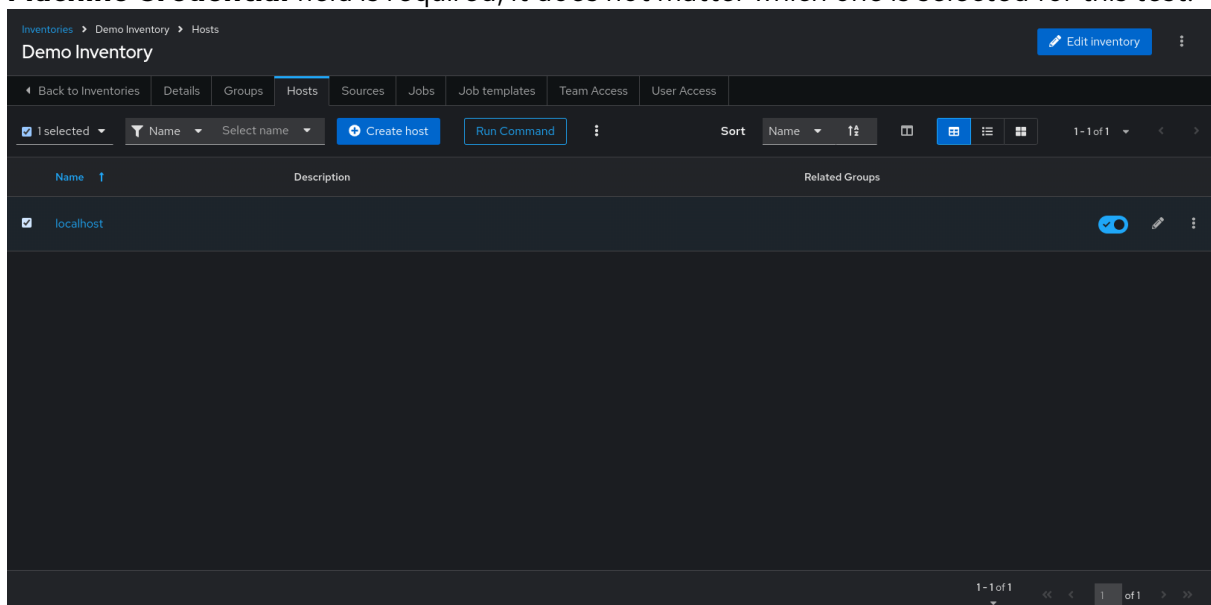


The screenshot shows the 'Create Inventory' form in the Ansible Tower UI. The form has several sections: 'Name' with an 'Enter name' input, 'Description' with an 'Enter description' input, and 'Organization' with a 'Select organization' dropdown. Below these are 'Instance Groups' with an 'Add instance groups' button and a search icon, and 'Labels' with a 'Select or create labels' dropdown. There is also a 'Variables' section with a text area and 'YAML'/'JSON' tabs. At the bottom, there are 'Options' including a checkbox for 'Prevent instance group fallback'.

2. Create the `localhost` host in the inventory with the following variables:

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

3. Launch an ad hoc job against the localhost using the *ping* or *setup* module. Even though the **Machine Credential** field is required, it does not matter which one is selected for this test:



The screenshot shows the 'Demo Inventory' Hosts page in the Ansible Tower UI. The page has a navigation bar with tabs for 'Back to Inventories', 'Details', 'Groups', 'Hosts' (selected), 'Sources', 'Jobs', 'Job templates', 'Team Access', and 'User Access'. Below the navigation bar is a table with columns for 'Name', 'Description', and 'Related Groups'. The table contains one host named 'localhost'. There are buttons for 'Create host' and 'Run Command'. The page also shows a 'Sort' dropdown and a '1-1 of 1' indicator.

Result

You can see in the **Jobs** details view that the container was reached successfully by using one of the ad hoc jobs.

If you have an OpenShift UI, you can see pods appear and disappear as they deploy and end. You can also use the CLI to perform a `get pod` operation on your namespace to watch these same events occurring in real-time.

Container capacity limits

When using container groups in automation controller, you can set capacity limits for the containers that run the jobs.

Capacity limits and quotas for containers are defined by objects in the Kubernetes API:

- To set limits on all pods within a given namespace, use the `LimitRange` object. For more information see the [Quotas and Limit Ranges](#) section of the OpenShift documentation.
- To set limits directly on the pod definition launched by automation controller, see [Customizing the pod specification](#) and the [Compute Resources](#) section of the OpenShift documentation.

NOTE:

Container groups do not use the capacity algorithm that normal nodes use. You need to set the number of forks at the job template level. If you configure forks in automation controller, that setting is passed along to the container.

Scale automation across your infrastructure with automation mesh

Learn how to plan an automation mesh deployment in your Ansible Automation Platform environment.

The subsequent sections:

- Explain the concepts that comprise automation mesh
- Provide examples on how you can design automation mesh topologies
- Include simple to complex topology examples that illustrate the various ways you can deploy automation mesh
- Contain information to help plan an automation mesh deployment in your operator-based Red Hat Ansible Automation Platform environment
- Covers the setting up of automation mesh on operator-based deployments, such as OpenShift Container Platform and Ansible Automation Platform on Amazon Web Services (AWS) and Microsoft Azure managed applications.

About automation mesh

Automation mesh is an overlay network intended to ease the distribution of work across a large and dispersed collection of workers through nodes that establish peer-to-peer connections with each other using existing networks.

Red Hat Ansible Automation Platform 2 replaces Ansible Tower and isolated nodes with Ansible Automation Platform and automation hub. Ansible Automation Platform provides the control plane for automation through its UI, RESTful API, RBAC, workflows and CI/CD integration, while automation mesh can be used for setting up, discovering, changing or modifying the nodes that form the control and execution layers.

Automation mesh is useful for:

- traversing difficult network topologies
- bringing execution capabilities (the machine running `ansible-playbook`) closer to your target hosts

The nodes (control, hop, and execution instances) are interconnected through a receptor mesh, forming a virtual mesh.

Automation mesh uses TLS encryption for communication, so traffic that traverses external networks (the internet or other) is encrypted in transit.

Automation mesh introduces:

- Dynamic cluster capacity that scales independently, enabling you to create, register, group, ungroup and deregister nodes with minimal downtime.
- Control and execution plane separation that enables you to scale playbook execution capacity independently from control plane capacity.
- Deployment choices that are resilient to latency, reconfigurable without outage, and that dynamically re-reroute to choose a different path when outages occur.
- Connectivity that includes bi-directional, multi-hopped mesh communication possibilities which are *Federal Information Processing Standards* (FIPS) compliant.

Control and execution planes

Automation mesh makes use of unique node types to create both the **control** and **execution** plane. Learn more about the control and execution plane and their node types before designing your automation mesh topology.

Control plane

The control plane contains the persistent services that manage and scale your automation environment. The specific set up of the control plane depends on your deployment model:

For VM environment mesh deployments

The **control plane** consists of hybrid and control nodes. Instances in the control plane run persistent automation controller services such as the the web server and task dispatcher, in addition to project updates, and management jobs.

Hybrid nodes

This is the default node type for control plane nodes, responsible for automation controller runtime functions like project updates, management jobs and `ansible-runner` task operations. Hybrid nodes are also used for automation execution.

Control nodes

Control nodes run project and inventory updates and system jobs, but not regular jobs. Execution capabilities are disabled on these nodes.

For managed cloud or operator environment mesh deployments

Instances in the control plane run persistent Ansible Automation Platform services such as the web server and task dispatcher, in addition to project updates, and management jobs. However, in the operator-based model, 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.

Execution plane

The **execution plane** consists of execution nodes that execute automation on behalf of the control plane and have no control functions. Hop nodes serve to communicate. Nodes in the **execution plane** only run user-space jobs, and may be geographically separated, with high latency, from the control plane.

Execution notes

Execution nodes run jobs under `ansible-runner` with `podman` isolation. This node type is similar to isolated nodes. This is the default node type for execution plane nodes.

Hop nodes

Similar to a jump host, hop nodes route traffic to other execution nodes. Hop nodes cannot execute automation.

Peers

Peer relationships are essential for defining the node-to-node connections that form the automation mesh. The method for defining peers depends on your specific deployment model:

For VM environment mesh deployments

You can define peers within the `[automationcontroller]` and `[execution_nodes]` groups or using the `[automationcontroller:vars]` or `[execution_nodes:vars]` groups

For managed cloud or operator environment mesh deployments

Peers are defined through the UI for individual instances.

Related information

[Define automation mesh node types](#)

Define automation mesh node types

The examples in this section demonstrate how to set the node type for the hosts in your inventory file.

IMPORTANT:

For a container-based installation of Ansible Automation Platform, replace `node_type` with `receptor_type`.

You can set the `node_type` for single nodes in the control plane or execution plane inventory groups. To define the node type for an entire group of nodes, set the `node_type` in the `vars` stanza for the group.

- The permitted values for `node_type` in the control plane `[automationcontroller]` group are `hybrid` (default) and `control`.
- The permitted values for `node_type` in the `[execution_nodes]` group are `execution` (default) and `hop`.

Hybrid nodes

The following inventory consists of a single hybrid node in the control plane:

```
[automationcontroller]
control-plane-1.example.com
```

Control nodes

The following inventory consists of a single control node in the control plane:

```
[automationcontroller]
control-plane-1.example.com node_type=control
```

If you set `node_type` to `control` in the `vars` stanza for the control plane nodes, then all of the nodes in control plane are control nodes.

```
[automationcontroller]
control-plane-1.example.com

[automationcontroller:vars]
node_type=control
```

Execution nodes

The following stanza defines a single execution node in the execution plane:

```
[execution_nodes]
execution-plane-1.example.com
```

Hop nodes

The following stanza defines a single hop node and an execution node in the execution plane. The `node_type` variable is set for every individual node.

```
[execution_nodes]
execution-plane-1.example.com node_type=hop
execution-plane-2.example.com
```

If you want to set the `node_type` at the group level, you must create separate groups for the execution nodes and the hop nodes.

```
[execution_nodes]
execution-plane-1.example.com
execution-plane-2.example.com

[execution_group]
execution-plane-2.example.com

[execution_group:vars]
node_type=execution

[hop_group]
execution-plane-1.example.com

[hop_group:vars]
node_type=hop
```

Peer connections

IMPORTANT:

For a container-based installation of Ansible Automation Platform, use the `receptor_peers=` variable instead of `peers=`.

The value of `receptor_peers` must be a comma-separated list of hostnames. Do not use inventory group names. For more information, see [Adding execution nodes](#).

Create node-to-node connections using the `peers=` host variable. The following example connects `control-plane-1.example.com` to `execution-node-1.example.com` and `execution-node-1.example.com` to `execution-node-2.example.com`:

```
[automationcontroller]
control-plane-1.example.com peers=execution-node-1.example.com

[automationcontroller:vars]
node_type=control

[execution_nodes]
execution-node-1.example.com peers=execution-node-2.example.com
execution-node-2.example.com
```

See the example automation mesh topologies in this section for more examples of how to implement mesh nodes.

Scale with automation mesh in an operator environment

Scaling your automation mesh is available on OpenShift deployments of Red Hat Ansible Automation Platform and is possible through adding or removing nodes from your cluster dynamically, using the **Instances** resource of the Ansible Automation Platform UI, without running the installation script.

Instances serve as nodes in your mesh topology. Automation mesh enables you to extend the footprint of your automation. The location where you launch a job can be different from the location where the ansible-playbook runs.

To manage instances from the Ansible Automation Platform UI, you must have System Administrator or System Auditor permissions.

In general, the more processor cores (CPU) and memory (RAM) a node has, the more jobs that can be scheduled to run on that node at once.

Prerequisites

The automation mesh is dependent on hop and execution nodes running on *Red Hat Enterprise Linux* (RHEL). Your Red Hat Ansible Automation Platform subscription grants you ten Red Hat Enterprise Linux licenses that can be used for running components of Ansible Automation Platform.

For more information about Red Hat Enterprise Linux subscriptions, see **Registering the system and managing subscriptions** in the Red Hat Enterprise Linux documentation.

The following steps prepare the RHEL instances for deployment of the automation mesh.

1. You require a Red Hat Enterprise Linux operating system. Each node in the mesh requires a static IP address, or a resolvable DNS hostname that Ansible Automation Platform can access.
2. Ensure that you have the minimum requirements for the RHEL virtual machine before proceeding. For more information, see the **System requirements**.
3. Deploy the RHEL instances within the remote networks where communication is required. For information about creating virtual machines, see **Creating Virtual Machines** in the *Red Hat Enterprise Linux* documentation. Remember to scale the capacity of your virtual machines sufficiently so that your proposed tasks can run on them.
 - RHEL ISOs can be obtained from access.redhat.com.
 - RHEL cloud images can be built using Image Builder from console.redhat.com.

Related information

[How job capacity is determined and impacts job runs](#)

[Registering the system and managing subscriptions](#)

[Red Hat Ansible Automation Platform system requirements](#)

[Creating Virtual Machines](#)

Set up VMs for use in mesh

Prepare Red Hat Enterprise Linux virtual machines by setting up SSH access and enabling repositories. Deploying these VMs as nodes helps expand the execution capacity of your automation mesh.

Procedure

1. SSH into each of the RHEL instances and perform the following steps. Depending on your network access and controls, SSH proxies or other access models might be required.

Use the following command:

```
ssh [username]@[host_ip_address]
```

For example, for an Ansible Automation Platform instance running on Amazon Web Services.

```
ssh ec2-user@10.0.0.6
```

2. Create or copy an SSH key that can be used to connect from the hop node to the execution node in later steps. This can be a temporary key used just for the automation mesh configuration, or a long-lived key. The SSH user and key are used in later steps.
3. Enable your RHEL subscription with `baseos` and `appstream` repositories. Ansible Automation Platform RPM repositories are only available through subscription-manager, not the *Red Hat Update Infrastructure* (RHUI). If you attempt to use any other Linux footprint, including RHEL with RHUI, this causes errors.

```
sudo subscription-manager register --auto-attach
```

If Simple Content Access is enabled for your account, use:

```
sudo subscription-manager register
```

For more information about Simple Content Access, see [Getting started with simple content access](#).

4. Enable Ansible Automation Platform subscriptions and the proper Red Hat Ansible Automation Platform channel:

For RHEL 8

```
# subscription-manager repos --enable ansible-automation-platform-2.5-for-rhel-8-x86_64-rpms
```

For RHEL 9

```
# subscription-manager repos --enable ansible-automation-platform-2.5-for-rhel-9-x86_64-rpms
```

For ARM

```
# subscription-manager repos --enable ansible-automation-platform-2.5-for-rhel-aarch64-rpms
```

5. Ensure the packages are up to date:

```
sudo dnf upgrade -y
```

6. Install the ansible-core packages on the machine where the downloaded bundle is to run:

```
sudo dnf install -y ansible-core
```

NOTE:

Ansible core is required on the machine that runs the automation mesh configuration bundle playbooks. This document assumes that happens on the execution node. However, this step can be omitted if you run the playbook from a different machine. You cannot run directly from the control node, this is not currently supported, but future support expects that the control node has direct connectivity to the execution node.

Add execution or hop nodes to scale your infrastructure

To expand job capacity, create a standalone **execution node** that can be added to run alongside a deployment of automation controller. These execution nodes are not part of the automation controller Kubernetes cluster.

The control nodes run in the cluster connect and submit work to the execution nodes through Receptor.

These execution nodes are registered in automation controller as type `execution` instances, meaning they are only used to run jobs, not dispatch work or handle web requests as control nodes do.

IMPORTANT:

When creating an execution node, make sure the system time zone on the execution node matches `settings.TIME_ZONE` (default is 'UTC') on automation controller. Fact caching relies on comparing modified times of artifact files, and these modified times are not time zone-aware. Therefore, it is critical that the timezones of the execution nodes match automation controller's time zone setting.

Hop nodes can be added to sit between the control plane of automation controller and standalone execution nodes. These hop nodes are not part of the Kubernetes cluster and are registered in automation controller as an instance of type `hop`, meaning they only handle inbound and outbound traffic for otherwise unreachable nodes in different or more strict networks.

The following procedure demonstrates how to set the node type for the hosts.

NOTE:

By default, Red Hat Ansible Automation Platform Service on AWS includes two hop nodes that you can peer execution nodes to.

Procedure

1. From the navigation panel, select **Automation Execution > Infrastructure > Instances**.
2. On the **Instances** list page, click **Add instance**. The **Add Instance** window opens.

The screenshot shows the 'Add instance' form. At the top, there's a breadcrumb 'Instances > Add instance' and the title 'Add instance'. Below this, there are three input fields: 'Host name' with a placeholder 'Enter a host name', 'Instance state' with a dropdown menu showing 'installed', and 'Listener port' with a placeholder 'Enter a listener port'. Below these is a dropdown for 'Instance type' with 'Execution' selected. At the bottom, there's an 'Options' section with three checkboxes: 'Enable instance', 'Managed by policy', and 'Peers from control nodes', all of which are currently unchecked.

An instance requires the following attributes:

- **Host name:** (required) Enter a fully qualified domain name (public DNS) or IP address for your instance. This field is equivalent to `hostname` for installer-based deployments.

NOTE:

If the instance uses private DNS that cannot be resolved from the control cluster, DNS lookup routing fails, and the generated SSL/TLS certificates is invalid. Use the IP address instead.

- Optional: **Description:** Enter a description for the instance.
- **Instance state:** This field is auto-populated, indicating that it is being installed, and cannot be modified.
- **Listener port:** This port is used for the receptor to listen on for incoming connections. You can set the port to one that is appropriate for your configuration. This field is equivalent to `listener_port` in the API. The default value is 27199, though you can set your own port value.
- **Instance type:** Only `execution` and `hop` nodes can be created. Operator based deployments do not support Control or Hybrid nodes.
Options:
 - **Enable instance:** Check this box to make it available for jobs to run on an execution node.

- Check the **Managed by policy** box to enable policy to dictate how the instance is assigned.
 - **Peers from control nodes:**
 - If you are configuring a hop node:
 - If the hop node needs to have requests pushed directly from automation controller, then check the **Peers from Control** box.
 - If the hop node is peered to another hop node, then make sure **Peers from Control** is not checked.
 - If you are configuring an execution node:
 - If the execution node needs to have requests pushed directly from automation controller, then check the **Peers from Control** box.
 - If the execution node is peered to a hop node, then make sure that **Peers from Control** is not checked.
3. Click **Associate peers**.
 4. To view a graphical representation of your updated topology, see [Topology view](#).

NOTE:

Complete the following steps from any computer that has SSH access to the newly created instance.

5. Click the  icon next to **Download Bundle** to download the tar file that includes this new instance and the files necessary to install the created node into the automation mesh.

The install bundle has TLS certificates and keys, a certificate authority, and a proper Receptor configuration file.

```
receptor-ca.crt
work-public-key.pem
receptor.key
install_receptor.yml
inventory.yml
group_vars/all.yml
requirements.yml
```

6. Extract the downloaded `tar.gz` Install Bundle from the location where you downloaded it. To ensure that these files are in the correct location on the remote machine, the install bundle includes the `install_receptor.yml` playbook.
7. Before running the `ansible-playbook` command, edit the following fields in the `inventory.yml` file:

```
all:
  hosts:
    remote-execution:
      ansible_host: localhost # change to the mesh node host name
      ansible_user: <username> # user provided
      ansible_ssh_private_key_file: ~/.ssh/<id_rsa>
```

- Ensure `ansible_host` is set to the IP address or DNS of the node.
- Set `ansible_user` to the username running the installation.
- Set `ansible_ssh_private_key_file` to contain the filename of the private key used to connect to the instance.
- The content of the `inventory.yml` file serves as a template and has variables for roles that are applied during the installation and configuration of a receptor node in a mesh topology. You can change some of the other fields, or replace the file in its entirety for advanced scenarios. For more information, see [Role Variables](#).

8. For a node that uses a private DNS, add the following line to `inventory.yml`:

```
ansible_ssh_common_args: <your ssh ProxyCommand setting>
```

This instructs the `install-receptor.yml` playbook to use the proxy command to connect through the local DNS node to the private node.

9. When the attributes are configured, click **Save**. The **Details** page of the created instance opens.
10. Save the file to continue.
11. The system that is going to run the install bundle to setup the remote node and run `ansible-playbook` requires the `ansible.receptor` collection to be installed:

```
ansible-galaxy collection install ansible.receptor
```

or

```
ansible-galaxy install -r requirements.yml
```

- Installing the receptor collection dependency from the `requirements.yml` file consistently retrieves the receptor version specified there. Additionally, it retrieves any other collection dependencies that might be needed in the future.

- Install the receptor collection on all nodes where your playbook will run, otherwise an error occurs.
12. If `receptor_listener_port` is defined, the machine also requires an available open port on which to establish inbound TCP connections, for example, 27199. Run the following command to open port 27199 for receptor communication (Make sure you have port 27199 open in your firewall):

```
sudo firewall-cmd --permanent --zone=public --add-port=27199/tcp
```

13. Run the following playbook on the machine where you want to update your automation mesh:

```
ansible-playbook -i inventory.yml install_receptor.yml
```

NOTE:

OpenSSL is required for this playbook. You can install it by running the following command:

```
openssl -v
```

If it returns then a version OpenSSL is installed. Otherwise you need to install OpenSSL with:

```
sudo dnf install -y openssl
```

After this playbook runs, your automation mesh is configured.

Instances 🔍

Name		Add		Remove		Run health check		1 - 3 of 3	
Name	Status	Node Type	Capacity Adjustment	Used Capacity	Actions				
☐ awx-mesh-ingress-1	Ready	hop							
☐ awx-task-65d6d96987-mgn9j	Ready	control	CPU 16 forks 156 RAM 156	Used capacity 0%	Enabled				
☐ ec2-35-87-18-213.us-west-2.compute.amazonaws.com	Ready	execution	CPU 4 forks 7 RAM 7	Used capacity 0%	Enabled				

1 - 3 of 3 items << < 1 of 1 page > >>

NOTE:

It might be the case that some servers do not listen on the receptor port (the default is 27199)

Suppose you have a Control plane with nodes A, B, and C

The following is a peering set up for three controller nodes:

Controller node A → Controller node B

Controller node A → Controller node C

Controller node B → Controller node C

You can force the listener by setting

```
receptor_listener=True
```

However, a connection Controller B → A is likely to be rejected as that connection already exists.

This means that nothing connects to Controller A as Controller A is creating the connections to the other nodes, and the following command does not return anything on Controller A:

```
[root@controller1 ~]# ss -ntlp | grep 27199 [root@controller1 ~]#
```

The RPM installer creates a strongly connected peering between the control plane nodes with a least privileged approach and opens the TCP listener only on those nodes where it is required. All the receptor connections are bidirectional, so when the connection is created, the receptor can communicate in both directions.

To remove an instance from the mesh, see [Removing instances](#).

Create an instance group

You can create instance groups with Automation controller to organize and manage your instances.

Use the following procedure to create a new instance group.

Procedure

1. From the navigation panel, select **Automation Execution > Infrastructure > Instance Groups**.
2. Click **Create group** and select **Create instance group** from the list.
3. Enter the appropriate details into the following fields:
 - **Name:** Names must be unique and must not be named "controller".
 - **Policy instance minimum:** Enter the minimum number of instances to automatically assign to this group when new instances come online.
 - **Policy instance percentage:** Use the slider to select a minimum percentage of instances to automatically assign to this group when new instances come online.

NOTE:

Policy instance fields are not required to create a new instance group. If you do not specify values, then the **Policy instance minimum** and **Policy instance percentage** default to 0.

- **Max concurrent jobs:** Specify the maximum number of forks that can be run for any given job.
- **Max forks:** Specify the maximum number of concurrent jobs that can be run for any given job.

NOTE:

The default value of 0 for **Max concurrent jobs** and **Max forks** denotes no limit. For more information, see [Instance group capacity limits](#).

4. Click **Create instance group**, or, if you have edited an existing Instance Group click **Save instance group**

Next steps

When you have successfully created the instance group the **Details** tab of the newly created instance group enables you to review and edit your instance group information.

You can also edit **Instances** and review **Jobs** associated with this instance group:

Instance groups > Test group > Details		
Test group Edit instance group		
Back to Instance Groups Details Instances Jobs		
Name	Type	Policy Instance Minimum ⓘ
Test group	Instance Group	0
Policy instance percentage ⓘ	Max concurrent jobs ⓘ	Max forks ⓘ
0%	2	6
Used Capacity	Created	Last modified
0	5/22/2024, 10:25:01 AM	5/22/2024, 10:25:01 AM

Associate instances to an instance group

Learn how to associate instances to an instance group in automation controller.

Procedure

1. Select the **Instances** tab on the **Details** page of an Instance Group.
2. Click **Associate instance**.
3. Click the checkbox next to one or more available instances from the list to select the instances you want to associate with the instance group and click **Confirm**

Run jobs on execution nodes

You must specify where jobs run from, or they default to running in the control cluster. To do this, set up a Job Template. For more information about Job Templates, see [Standardize and streamline automation with automation job templates](#).

Procedure

1. The **Templates** list view shows job templates that are currently available. From this screen you can launch 🚀, edit ✎, and duplicate 📄 a workflow job template.
2. Select the job you want and click the 🚀 icon.
3. Select the **Instance Group** on which you want to run the job. Note that a System Administrator must grant you or your team permissions to be able to use an instance group in a job template. If you select multiple jobs templates, the order in which you select them sets the execution precedence.
4. Click **Next**.
5. Click **Launch**.

Set up mesh ingress for environments that forbid inbound connections

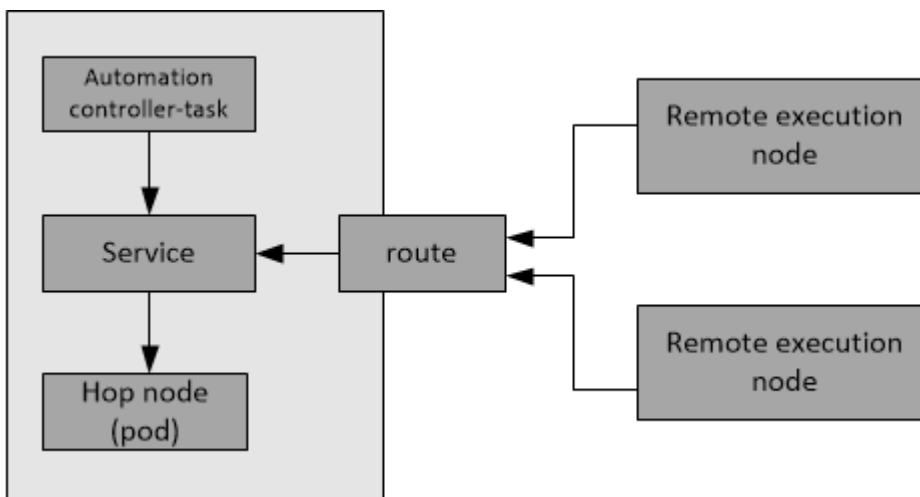
If your network restricts inbound connections, using a hop node peered to the control plane can cause issues, as it requires a defined 'listener_port'. Instead, you can use *mesh ingress* as an alternative method for setting up your automation mesh.

Before you begin

- Create node instances within the remote networks for execution nodes in the automation mesh.

Use the following procedure to set up mesh nodes.

When you instantiate mesh ingress you set up a pod, or receptor hop node inside the kubernetes control cluster, registered to the database through the operator. It also creates a service, and a route URL that is used by the control plane to connect to the hop node, and automation controller.



Procedure

1. Create a YAML file (in this case named `oc_meshingress.yml`) to set up the mesh ingress node.

Your file should resemble the following:

```

apiVersion: automationcontroller.ansible.com/v1alpha1
kind: AutomationControllerMeshIngress
metadata:
  name:
  namespace:
spec:
  deployment_name: aap-controller

```

Where:

- **apiVersion:** defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and might reject unrecognized values. This value is static.
- **kind:** Is the type of node to create.
Use the value `AutomationControllerMeshIngress`.
`AutomationControllerMeshIngress` controls the deployment and configuration of mesh ingress on automation controller.
- **name:** enter a name for the mesh ingress node.
- **namespace:** enter a name for the Kubernetes namespace to deploy the mesh ingress into.
This must be in the same namespace as the automation controller that the mesh is connected to.
- **deployment_name:** is the automation controller instance that this mesh ingress is attached to. Provide the name of your automation controller instance.

2. Apply this YAML file using:

```
oc apply -f oc_meshingress.yml
```

Run this playbook to creates the `AutomationControllerMeshIngress` resource. The operator creates a hop node in automation controller with the `name` you supplied.

3. When the MeshIngress instance has been created, it appears in the Instances page.

IMPORTANT:

Any instance that is to function as a remote execution node in "pull" mode need to be created after this procedure and must be configured as follows:

```
instance type: Execution
listener port: keep empty
options:
  Enable instance: checked
  Managed by Policy: as needed
  Peers from control nodes: unchecked (this one is important)
```

4. Associate this new instance with the hop node you created using the procedure in this paragraph
5. Download the tarball.

NOTE:

Association with the hop node must be done before creating the tarball.

Create a pull secret to run the default execution environment on remote nodes

If you are using the default execution environment provided with automation controller to run on remote execution nodes, you must add a pull secret in automation controller that has the credential for pulling the execution environment image.

NOTE:

This does not apply to Ansible Automation Platform on Microsoft Azure.

Create a pull secret on the automation controller namespace and configure the `ee_pull_credentials_secret` parameter in the Operator as follows:

Procedure

1. Create a secret using the following command:

```
oc create secret generic ee-pull-secret \
  --from-literal=username=<username> \
  --from-literal=password=<password> \
  --from-literal=url=registry.redhat.io
```

2. Add `ee_pull_credentials_secret` and `ee-pull-secret` to the specification by editing the deployment specification:

```
oc edit automationcontrollers aap-controller-o yaml
```

and add the following:

```
spec
  ee_pull_credentials_secret=ee-pull-secret
```

3. To manage instances from the automation controller UI, you must have System Administrator or System Auditor permissions.

Use custom signed certificates in managed cloud and operator environments

Execution nodes verify incoming connections by ensuring the x509 certificate was issued by a trusted Certificate Authority (CA). You might want to provide your own CA for this validation. If no CA is provided, Controller Operator generates a self-signed CA during installation.

The control nodes on the Kubernetes cluster communicate with execution nodes through mutual TLS/TCP connections, running using receptor. Controller Operator generates a self-signed CA during installation by using OpenSSL.

Procedure

1. If custom `ca.crt` and `ca.key` are stored locally, run the following:

```
kubectl create secret tls controller-demo-12345-receptor-ca \ --cert=/path/to/ca.crt --key=/path/to/ca.key
```

The secret should be named `<Controller Custom Resource name>-receptor-ca`. In this example the Controller CR name is `controller-demo-12345`.

Replace `controller-demo-12345` with your Controller Custom Resource name.

2. If this secret is created after automation controller is deployed, run the following to restart the deployment:

```
kubectrl rollout restart deployment controller-demo-12345
```

WARNING:

Changing the receptor CA breaks connections to any existing execution nodes. These nodes enter an unavailable state, and jobs cannot run on them. You must download and re-run the install bundle for each execution node. This replaces the TLS certificate files with those signed by the new CA. The execution nodes then appear in a ready state after a few minutes.

Remove instances

From the **Instances** page, you can add, remove or run health checks on your nodes.

NOTE:

You must follow the procedures for installing RHEL packages for any additional nodes you create. If you peer this additional node to an existing hop node, you must also install the Install Bundle on each node.

Use the check boxes next to an instance to select it to remove it, or run a health check against it.

NOTE:

- If a node is removed using the UI, then the node is "removed" and does not show a status. If you delete the VM of the node before it is removed in the UI, it will show an error.
- You only need to reinstall the Install Bundle if the topology changes the communication pattern, that is, hop nodes change or you add nodes.

When a button is disabled, you do not have permission for that particular action. Contact your Administrator to grant you the required level of access.

If you are able to remove an instance, you receive a prompt for confirmation.

NOTE:

You can still remove an instance even if it is active and jobs are running on it. Automation controller waits for jobs running on this node to complete before removing it.

Upgrade receptors

A software update addresses any issues or bugs to provide a better experience of working with the technology. Anyone with administrative rights can update the receptor on an execution node.

Red Hat recommends performing updates to the receptor after any Ansible Automation Platform control plane updates. This ensures you are using the latest version. The best practice is to perform regular updates outside of any updates to the control plane.

Procedure

1. Check the current receptor version:

```
receptor --version
```

2. Update the receptor:

```
sudo dnf update ansible-runner receptor -y
```

NOTE:

To upgrade all packages (not just the receptor), use `dnf update`, then reboot with `reboot`.

3. Verify the installation. After the update is complete, check the receptor version again to verify the update:

```
receptor --version
```

4. Restart the receptor service:

```
sudo systemctl restart receptor
```

5. Ensure the receptor is working correctly and is properly connected to the controller or other nodes in the system.

View details about mesh nodes with topology viewer

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

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

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

Related information

[Scale with automation mesh in an operator environment](#)

Access the topology viewer

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

Procedure

1. From the navigation panel, select **Automation Execution > Infrastructure > Topology View**. The **Topology View** opens and displays a graphical representation of how each receptor node links together.
2. To adjust the zoom levels, or manipulate the graphic views, use the control icons: zoom-in (⊕), zoom-out (⊖), expand (⌵), and reset (↺) on the toolbar.
 You can also click and drag to pan around; and scroll using your mouse or trackpad to zoom. The fit-to-screen feature automatically scales the graphic to fit on the screen and repositions it in the center. It is particularly useful when you want to see a large mesh in its entirety.

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

3. Refer to the **Legend** to identify the type of nodes that are represented.

For VM-based installations, see [Control and execution planes](#).

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

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

The following link statuses are also shown in the Legend:

- **Established:** This is a link state that indicates a peer connection between nodes that are either ready, unavailable, or disabled.
 - **Adding:** This is a link state indicating a peer connection between nodes that were selected to be added to the mesh topology.
 - **Removing:** This is a link state indicating a peer connection between nodes that were selected to be removed from the topology.
4. Hover over a node and the connectors highlight to show its immediate connected nodes (peers) or click a node to retrieve details about it, such as its hostname, node type, and status.
 5. Click the link for instance hostname from the details displayed to be redirected to its **Details** page that provides more information about that node, most notably for information about an `Error` status, as in the following example.

You can use the **Details** page to remove the instance, run a health check on the instance on an as-needed basis, or unassign jobs from the instance. By default, jobs can be assigned to each node. However, you can disable it to exclude the node from having any jobs running on it.

Related information

[Scale with automation mesh in an operator environment](#)

Scale with automation mesh in a containerized or VM environment

Run automation on different execution nodes across your data center, clouds, or remote sites with automation mesh.

For RPM or Containerized VM-based Ansible Automation Platform you use an installation program to set up automation mesh or to upgrade automation mesh. To provide Ansible Automation Platform with details about the nodes, groups, and peer relationships in your mesh network, you define them in the inventory file in the installer bundle.

Related information

[Scale with automation mesh in an operator environment](#)

[Design patterns for automation mesh](#)

Edit the inventory file

You can use the Ansible Automation Platform installer inventory file to specify your installation scenario.

For an RPM installation

Procedure

RPM installed package

```
$ cd /opt/ansible-automation-platform/installer/
```

Bundled installer

```
$ cd ansible-automation-platform-setup-bundle-<latest-version>
```

Online installer

```
$ cd ansible-automation-platform-setup-<latest-version>
```

For online installations

For online installations

```
cd <path-to-source-file>/ansible-automation-platform-containerized-setup-  
<version_number>
```

For offline or bundled installations

```
cd <path-to-source-file>/ansible-automation-platform-containerized-setup-bundle-  
<version_number>-<arch_name>
```

1. Open the `inventory` file with a text editor.
2. Edit the `inventory` file parameters to specify your installation scenario.
For containerized installation, see [Configuring the inventory file](#)

Related information

Inventory file variables

Run the setup script

Run the installation program setup script after updating the inventory file with all the required parameters to begin the installation and configuration of the platform.

RPM installer

- Run the `setup.sh` script

```
$ sudo ./setup.sh
```

NOTE:

If you are running the setup as a non-root user with `sudo` privileges, you can use the following command:

```
$ ANSIBLE_BECOME_METHOD='sudo'
ANSIBLE_BECOME=True ./setup.sh
```

Result

Installation of Red Hat Ansible Automation Platform will begin.

Next steps

If you want to add additional nodes to your automation mesh after the initial setup, edit the inventory file to add the new node, then rerun the `setup.sh` script.

Containerized installer

For information on installing containerized Ansible Automation Platform, see [Installing containerized Ansible Automation Platform](#)

Import a CA certificate

A Certificate Authority (CA) verifies and signs individual node certificates in an automation mesh environment. You can provide your own CA by specifying the path to the certificate and the private RSA key file in the `inventory` file of your Red Hat Ansible Automation Platform installer.

NOTE: The Ansible Automation Platform installation program generates a CA if you do not provide one.

Procedure

1. Open the `inventory` file for editing.
2. Add the `mesh_ca_keyfile` variable and specify the full path to the private RSA key (`.key`).
3. Add the `mesh_ca_certfile` variable and specify the full path to the CA certificate file (`.cert`).
4. Save the changes to the inventory file.

```
[all:vars]
mesh_ca_keyfile=/tmp/<mesh_CA>.key
mesh_ca_certfile=/tmp/<mesh_CA>.cert
```

5. With the CA files added to the inventory file, run the installation program to apply the CA.

Result

This process copies the CA to the `/etc/receptor/tls/ca/` directory on each control and execution node on your mesh network.

Design patterns for mesh

The automation mesh topologies in this section provide examples you can use to design a mesh deployment in your environment. Examples range from a simple, hybrid node deployment to a complex pattern that deploys numerous automation controller instances, employing several execution and hop nodes.

IMPORTANT:

If you are creating a mesh similar to the following in a containerized environment:

- Replace the `node_type` variable with `receptor_type`
- Replace the `peers` variable with `receptor_peers`
- Replace inventory group names with explicit comma-separated lists of hostnames

The value of `receptor_peers` must be a comma-separated list of hostnames. Do not use inventory group names. For more information, see [Adding execution nodes](#) below.

Prerequisites

- You reviewed conceptual information on node types and relationships.

NOTE:

The following examples include images that illustrate the mesh topology. The arrows in the images indicate the direction of peering. After peering is established, the connection between the nodes allows bidirectional communication.

Related information

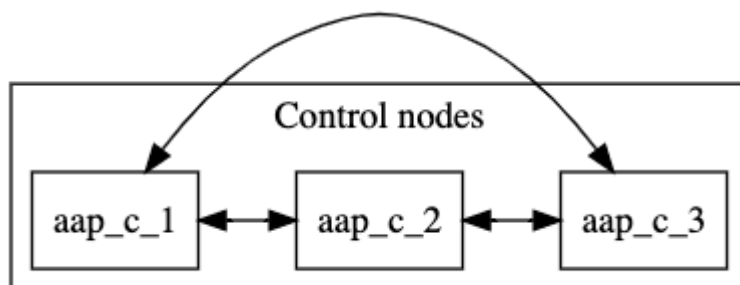
Add execution nodes

Multiple hybrid nodes inventory file example

This example inventory file deploys a control plane consisting of multiple hybrid nodes. The nodes in the control plane are automatically peered to one another.

```
[automationcontroller]
aap_c_1.example.com
aap_c_2.example.com
aap_c_3.example.com
```

The following image displays the topology of this mesh network.



The default `node_type` for nodes in the control plane is `hybrid`. You can explicitly set the `node_type` of individual nodes to `hybrid` in the `[automationcontroller group]`:

```
[automationcontroller]
aap_c_1.example.com node_type=hybrid
aap_c_2.example.com node_type=hybrid
aap_c_3.example.com node_type=hybrid
```

Alternatively, you can set the `node_type` of all nodes in the `[automationcontroller]` group. When you add new nodes to the control plane they are automatically set to hybrid nodes.

```
[automationcontroller]
aap_c_1.example.com
aap_c_2.example.com
aap_c_3.example.com

[automationcontroller:vars]
node_type=hybrid
```

If you think that you might add control nodes to your control plane in future, it is better to define a separate group for the hybrid nodes, and set the `node_type` for the group:

```
[automationcontroller]
aap_c_1.example.com
aap_c_2.example.com
aap_c_3.example.com

[hybrid_group]
aap_c_1.example.com
aap_c_2.example.com
aap_c_3.example.com

[hybrid_group:vars]
node_type=hybrid
```

Single node control plane with single execution node

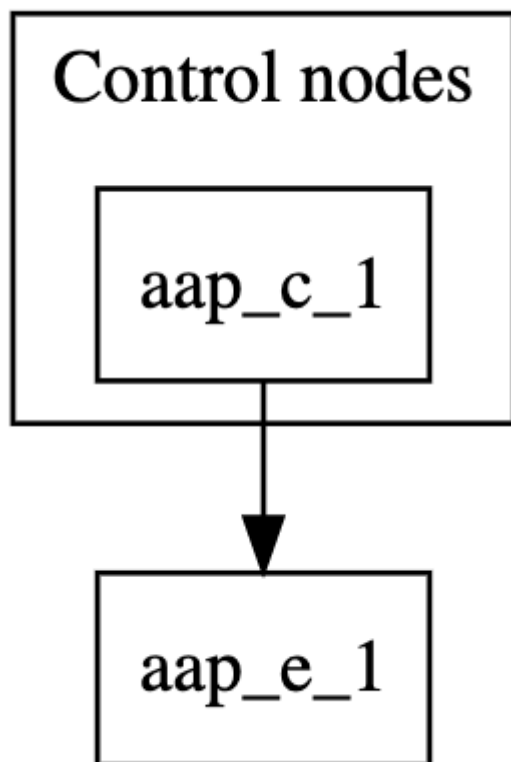
This example inventory file deploys a single-node control plane and establishes a peer relationship to an execution node.

```
[automationcontroller]
aap_c_1.example.com

[automationcontroller:vars]
node_type=control
peers=execution_nodes

[execution_nodes]
aap_e_1.example.com
```

The following image displays the topology of this mesh network.



The `[automationcontroller]` stanza defines the control nodes. If you add a new node to the `automationcontroller` group, it will automatically peer with the `aap_c_1.example.com` node.

The `[automationcontroller:vars]` stanza sets the node type to `control` for all nodes in the control plane and defines how the nodes peer to the execution nodes:

- If you add a new node to the `execution_nodes` group, the control plane nodes automatically peer to it.
- If you add a new node to the `automationcontroller` group, the node type is set to `control`.

The `[execution_nodes]` stanza lists all the execution and hop nodes in the inventory. The default node type is `execution`. You can specify the node type for an individual node:

```
[execution_nodes]
aap_e_1.example.com node_type=execution
```

Alternatively, you can set the `node_type` of all execution nodes in the `[execution_nodes]` group. When you add new nodes to the group, they are automatically set to execution nodes.

```
[execution_nodes]
aap_e_1.example.com

[execution_nodes:vars]
node_type=execution
```

If you plan to add hop nodes to your inventory in future, it is better to define a separate group for the execution nodes, and set the `node_type` for the group:

```
[execution_nodes]
aap_e_1.example.com

[local_execution_group]
aap_e_1.example.com

[local_execution_group:vars]
node_type=execution
```

Minimum resilient configuration

This inventory deploys a control plane with two control nodes and two execution nodes. All control nodes peer automatically. Because each control node connects to every node in the **execution_nodes** group, the configuration is resilient.

The capacity algorithm determines which control node is chosen when a job is launched. Refer to [Automation controller capacity determination and job impact](#) in Configuring automation execution for more information.

The following inventory file defines this configuration.

```
[automationcontroller]
aap_c_1.example.com
aap_c_2.example.com

[automationcontroller:vars]
node_type=control
peers=execution_nodes

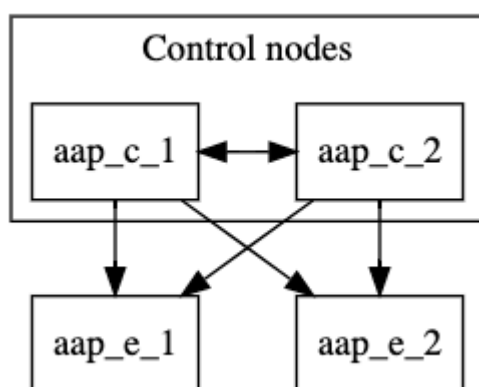
[execution_nodes]
aap_e_1.example.com
aap_e_2.example.com
```

The `[automationcontroller]` stanza defines the control nodes. All nodes in the control plane are peered to one another. If you add a new node to the `automationcontroller` group, it will automatically peer with the original nodes.

The `[automationcontroller:vars]` stanza sets the node type to `control` for all nodes in the control plane and defines how the nodes peer to the execution nodes:

- If you add a new node to the `execution_nodes` group, the control plane nodes automatically peer to it.
- If you add a new node to the `automationcontroller` group, the node type is set to `control`.

The following image displays the topology of this mesh network.



Segregated local and remote execution configuration

This configuration adds a hop node and a remote execution node to the resilient configuration. The remote execution node is reachable from the hop node.

You can use this setup if you are setting up execution nodes in a remote location, or if you need to run automation in a DMZ network.

```
[automationcontroller]
aap_c_1.example.com
aap_c_2.example.com

[automationcontroller:vars]
node_type=control
peers=instance_group_local

[execution_nodes]
aap_e_1.example.com
aap_e_2.example.com
aap_h_1.example.com node_type=hop
aap_e_3.example.com

[instance_group_local]
aap_e_1.example.com
aap_e_2.example.com

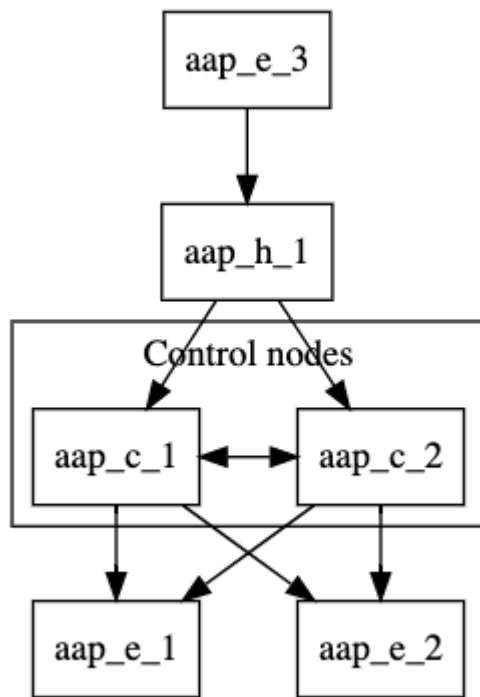
[hop]
aap_h_1.example.com

[hop:vars]
peers=automationcontroller

[instance_group_remote]
aap_e_3.example.com

[instance_group_remote:vars]
peers=hop
```

The following image displays the topology of this mesh network.



The `[automationcontroller:vars]` stanza sets the node types for all nodes in the control plane and defines how the control nodes peer to the local execution nodes:

- All nodes in the control plane are automatically peered to one another.
- All nodes in the control plane are peered with all local execution nodes.

If the name of a group of nodes begins with `instance_group_`, the installer recognises it as an instance group and adds it to the Ansible Automation Platform user interface.

Multi-hopped execution node

In this configuration, resilient controller nodes are peered with resilient local execution nodes. Resilient local hop nodes are peered with the controller nodes. A remote execution node and a remote hop node are peered with the local hop nodes.

You can use this setup if you need to run automation in a DMZ network from a remote network.

```

[automationcontroller]
aap_c_1.example.com
aap_c_2.example.com
aap_c_3.example.com

[automationcontroller:vars]
node_type=control
peers=instance_group_local

```

```

[execution_nodes]
aap_e_1.example.com
aap_e_2.example.com
aap_e_3.example.com
aap_e_4.example.com
aap_h_1.example.com node_type=hop
aap_h_2.example.com node_type=hop
aap_h_3.example.com node_type=hop

[instance_group_local]
aap_e_1.example.com
aap_e_2.example.com

[instance_group_remote]
aap_e_3.example.com

[instance_group_remote:vars]
peers=local_hop

[instance_group_multi_hop_remote]
aap_e_4.example.com

[instance_group_multi_hop_remote:vars]
peers=remote_multi_hop

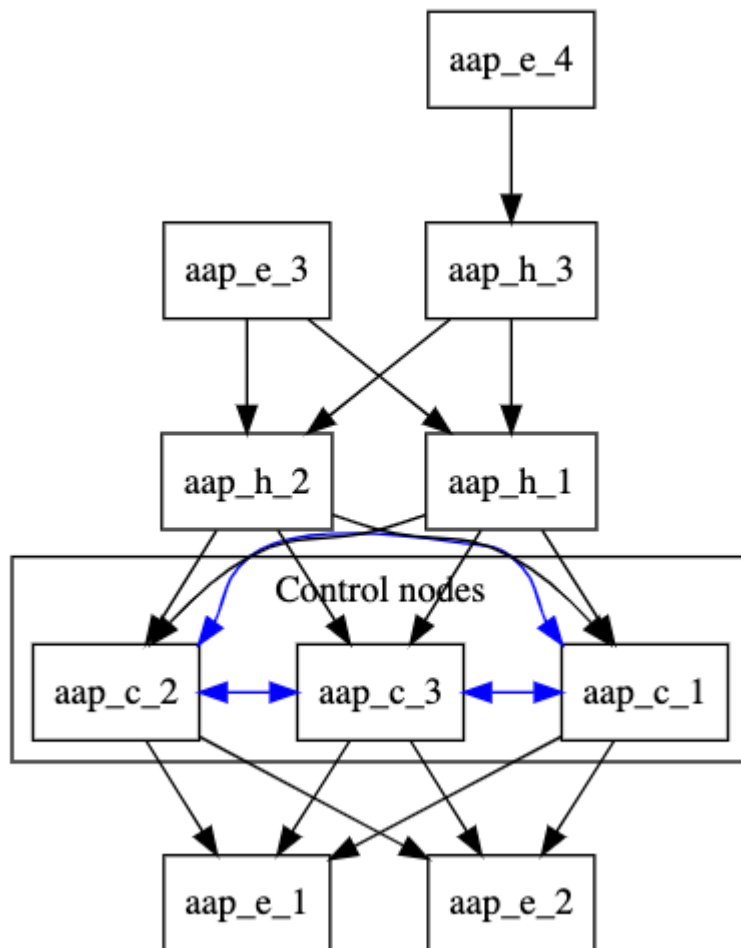
[local_hop]
aap_h_1.example.com
aap_h_2.example.com

[local_hop:vars]
peers=automationcontroller

[remote_multi_hop]
aap_h_3 peers=local_hop

```

The following image displays the topology of this mesh network.



The `[automationcontroller:vars]` stanza sets the node types for all nodes in the control plane and defines how the control nodes peer to the local execution nodes:

- All nodes in the control plane are automatically peered to one another.
- All nodes in the control plane are peered with all local execution nodes.

The `[local_hop:vars]` stanza peers all nodes in the `[local_hop]` group with all the control nodes.

If the name of a group of nodes begins with `instance_group_`, the installer recognises it as an instance group and adds it to the Ansible Automation Platform user interface.

Outbound only connections to controller nodes

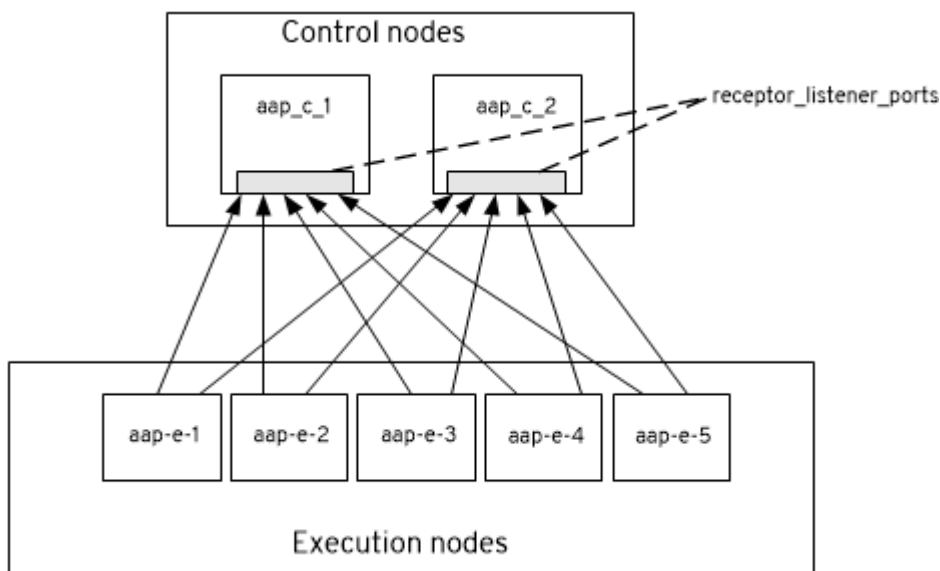
This example inventory file deploys a control plane consisting of two control nodes, and several execution nodes. Only outbound connections are allowed to the controller nodes. All nodes in the 'execution_nodes' group are peered with all nodes in the controller plane.

```
[automationcontroller]
controller-[1:2].example.com

[execution_nodes]
execution-[1:5].example.com

[execution_nodes:vars]
# connection is established *from* the execution nodes *to* the automationcontroller
peers=automationcontroller
```

The following image displays the topology of this mesh network.



Back up and restore your containerized deployment

Protecting your Ansible Automation Platform containerized deployment ensures that you can recover from system failures, data corruption, or configuration errors. Regular backups minimize downtime and protect your configurations and data.

Implementing backup and restore procedures helps you to:

- **Protect critical automation assets:** Safeguard your data including configurations, inventory, credentials, and job history from unexpected failures or data loss.
- **Enable disaster recovery:** Restore your environment quickly after hardware failures or other unexpected events to maintain business continuity.
- **Support testing and development:** Create copies of your environment for testing upgrades or configuration changes without affecting production systems.

Back up containerized Ansible Automation Platform

Perform a backup of your container-based installation of Ansible Automation Platform.

Before you begin

- You have logged in to the Red Hat Enterprise Linux host as your dedicated non-root user.

NOTE:

- When backing up Ansible Automation Platform, use the installation program that matches your currently installed version of Ansible Automation Platform.
- Backup functionality only works with the PostgreSQL versions supported by your current Ansible Automation Platform version.
- Backup and restore for content stored in Azure Blob Storage or Amazon S3 must be handled through the vendor portals, as each vendor provides their own backup solutions.

Procedure

1. Go to the Red Hat Ansible Automation Platform installation directory on your Red Hat Enterprise Linux host.
2. To control compression of the backup artifacts before they are sent to the host running the backup operation, you can use the following variables in your inventory file:
 - a. For control of compression for filesystem related backup files:

```
# Global control of compression for filesystem backup files
use_archive_compression=true

# Component-level control of compression for filesystem backup files
#controller_use_archive_compression=true
#eda_use_archive_compression=true
#gateway_use_archive_compression=true
#hub_use_archive_compression=true
#pcp_use_archive_compression=true
#postgresql_use_archive_compression=true
#receptor_use_archive_compression=true
#redis_use_archive_compression=true
```

b. For control of compression for database related backup files:

```
# Global control of compression for database backup files
use_db_compression=true

# Component-level control of compression for database backup files
#controller_use_db_compression=true
#eda_use_db_compression=true
#hub_use_db_compression=true
#gateway_use_db_compression=true
```

3. Run the `backup` playbook:

```
$ ansible-playbook -i <path_to_inventory>
  ansible.containerized_installer.backup
```

The backup process creates archives of the following data:

- PostgreSQL databases
- Configuration files
- Data files

Next steps

To customize the backup process, you can use the following variables in your inventory file:

- Change the backup destination directory from the default `./backups` by using the `backup_dir` variable.
- Exclude paths that contain duplicated data, such as snapshot subdirectories, by using the `hub_data_path_exclude` variable.

For example, to exclude a `.snapshots` subdirectory from the backup, add the following to your inventory file:

```
hub_data_path_exclude=["*/.snapshots", "*/.snapshots/*"]
```

Alternatively, you can pass this variable at runtime by using the `-e` flag:

```
$ ansible-playbook -i inventory ansible.containerized_installer.backup -e
  hub_data_path_exclude=["*/.snapshots", "*/.snapshots/*"]
```

You can also define the exclusion patterns in a YAML extra variables file and pass it at runtime:

exclude_vars.yml

```
hub_data_path_exclude:  
  - "*/.snapshots/*"  
  - "*/.snapshots"
```

```
$ ansible-playbook -i inventory ansible.containerized_installer.backup -e  
@exclude_vars.yml
```

Restore containerized Ansible Automation Platform

Restore your container-based installation of Ansible Automation Platform from a backup, or to a different environment.

Before you begin

- You have logged in to the Red Hat Enterprise Linux host as your dedicated non-root user.
- You have a backup of your Ansible Automation Platform deployment. For more information, see [See "Back up containerized Ansible Automation Platform" on page 1120](#).
- If restoring to a different environment with the same hostnames, you have performed a fresh installation on the target environment with the same topology as the original (source) environment.
- You have ensured that the administrator credentials on the target environment match the administrator credentials from the source environment.

NOTE:

When restoring Ansible Automation Platform, use the latest installation program available at the time of the restore. For example, if you are restoring a backup taken from version `2.6-1`, use the latest `2.6-x` installation program available at the time of the restore.

Restore functionality only works with the PostgreSQL versions supported by your current Ansible Automation Platform version. For more information, see [System requirements](#).

Procedure

1. Go to the installation directory on your Red Hat Enterprise Linux host.

2. Perform the relevant restoration steps:

- If you are restoring to the same environment with the same hostnames, run the `restore` playbook:

```
$ ansible-playbook -i <path_to_inventory>
ansible.containerized_installer.restore
```

This restores the important data deployed by the containerized installer such as:

- PostgreSQL databases
- Configuration files
- Data files

By default, the backup directory is set to `./backups`. You can change this by using the `backup_dir` variable in your `inventory` file.

- If you are restoring to a different environment with different hostnames, perform the following additional steps before running the `restore` playbook:

IMPORTANT: Restoring to a different environment with different hostnames is not recommended and is intended only as a workaround.

- For each component, identify the backup file from the source environment that contains the PostgreSQL dump file.

For example:

```
$ cd ansible-automation-platform-containerized-setup-
<version_number>/backups
```

```
$ tar tvf gateway_env1-gateway-node1.tar.gz | grep db
```

```
-rw-r--r-- ansible/ansible 4850774 2025-06-30 11:05 aap/backups/
awx.db
```

- Copy the backup files from the source environment to the target environment.
- Rename the backup files on the target environment to reflect the new node names.

For example:

```
$ cd ansible-automation-platform-containerized-setup-
<version_number>/backups
```

```
$ mv gateway_env1-gateway-node1.tar.gz gateway_env2-gateway-
node1.tar.gz
```

- d. For enterprise topologies, ensure that the component backup file containing the `component.db` file is listed first in its group within the inventory file. For example:

```
$ cd ansible-automation-platform-containerized-setup-
<version_number>
```

```
$ ls backups/gateway*
```

```
gateway_env2-gateway-node1.tar.gz
gateway_env2-gateway-node2.tar.gz
```

```
$ tar tvf backups/gateway_env2-gateway-node1.tar.gz | grep db
```

```
-rw-r--r-- ansible/ansible 416687 2025-06-30 11:05 aap/backups/
gateway.db
```

```
$ tar tvf backups/gateway_env2-gateway-node2.tar.gz | grep db
```

```
$ vi inventory
```

```
[automationgateway]
env2-gateway-node1
env2-gateway-node2
```

Back up and restore in an OpenShift environment

To safeguard against unexpected data loss and application errors, it is critical that you perform periodic backups of your Red Hat Ansible Automation Platform deployment. In addition to data loss prevention, backups allow you to fall back to a different deployment state.

About backup and recovery

Red Hat recommends backing up deployments of Red Hat Ansible Automation Platform in your Red Hat OpenShift Container Platform environment to prevent data loss.

A backup resource of your Red Hat Ansible Automation Platform deployment includes the following:

- Custom deployment of specific values in the `spec` section of the Ansible Automation Platform custom resource object.
- Back up of the `postgresql` database.
- `secret_key`, `admin_password`, and `broadcast_websocket` secrets.
- Database configuration.

NOTE:

Be sure to secure your backup resources because they can include sensitive information.

Backup recommendations

Recovering from data loss requires that you plan for and create backup resources of your Red Hat Ansible Automation Platform deployments on a regular basis. At a minimum, Red Hat recommends backing up deployments of Red Hat Ansible Automation Platform under the following circumstances:

- Before upgrading your Red Hat Ansible Automation Platform deployments.
- Before upgrading your OpenShift cluster.
- Once per week. This is particularly important if your environment is configured for automatic upgrades.

Back up your Ansible Automation Platform deployment

Backing up your Red Hat Ansible Automation Platform deployment involves creating backup resources for your deployed instances.

Use the following procedures to create backup resources for your Red Hat Ansible Automation Platform deployment. We recommend taking backups before upgrading the Ansible Automation

Platform Operator. Take a backup regularly in case you want to restore the platform to a previous state.

Back up your Ansible Automation Platform deployment

Regularly backing up your **Ansible Automation Platform** deployment is vital to protect against data loss. When you back up the platform, the operator automatically backs up all enabled components, including automation controller, automation hub, and Event-Driven Ansible.

Before you begin

- You must be authenticated on OpenShift cluster.
- You have installed Ansible Automation Platform Operator on the cluster.
- You have deployed a **Ansible Automation Platform** instance using the Ansible Automation Platform Operator.

NOTE:

Ansible Automation Platform Operator creates a PersistentVolumeClaim (PVC) for your Ansible Automation Platform Backup automatically. You can use your own pre-created PVC by using the `backup_pvc` spec and specifying your PVC.

Procedure

1. Log in to Red Hat OpenShift Container Platform.
2. Navigate to **Operators > Installed Operators**.
3. Select your Ansible Automation Platform Operator deployment.
4. Go to your **All Instances** tab, and click **Create New**.
5. Select **Ansible Automation Platform Backup** from the list.

NOTE:

When creating the **Ansible Automation Platform Backup** resource it also creates backup resources for each of the nested components that are enabled.

6. In the **Name** field, enter a name for the backup.
7. In the **Deployment name** field, enter the name of the deployed Ansible Automation Platform instance being backed up. For example if your Ansible Automation Platform deployment must be backed up and the deployment name is `aap`, enter `'aap'` in the **Deployment name** field.

8. Click **Create**. This results in an **AnsibleAutomationPlatformBackup** resource similar to the following:

```
apiVersion: aap.ansible.com/v1alpha1
kind: AnsibleAutomationPlatformBackup
metadata:
  name: backup
  namespace: aap
spec:
  no_log: true
  deployment_name: aap
```

Result

To verify that your backup was successful you can:

1. Log in to Red Hat OpenShift Container Platform.
2. Navigate to **Operators > Installed Operators**.
3. Select your Ansible Automation Platform Operator deployment.
4. Click **All Instances**.

The **All Instances** page displays the main backup and the backups for each component with the name you specified when creating your backup resource. The status for the following instances must be either **Running** or **Successful**:

- AnsibleAutomationPlatformBackup
- AutomationControllerBackup
- EDABackup
- AutomationHubBackup

Define custom backup Persistent Volume Claims

Define a custom Persistent Volume Claims (PVCs) to control backup storage allocation for each Ansible Automation Platform component. Specify unique PVC names, storage classes, and volume sizes at both global and component levels to differentiate between backup runs.

Before you begin

- You have an active Red Hat Ansible Automation Platform deployment on OpenShift Container Platform.
- You have the `oc` CLI tool installed and cluster administrator access.

Procedure

1. Create a backup YAML file, for example `custom-pvc-backup.yaml` , that defines `backup_pvc` and `create_backup_pvc` parameters for each component:

```

apiVersion: aap.ansible.com/v1alpha1
kind: AnsibleAutomationPlatformBackup
metadata:
  name: aapbackup
spec:
  backup_pvc: custom-aap-backup-pvc (a)
  backup_storage_class: nfs-local-rwx
  backup_storage_requirements: 7Gi
  create_backup_pvc: true (b)
  deployment_name: aap
  controller: (c)
    backup_pvc: custom-controller-backup-pvc
    backup_resource_requirements:
      limits:
        cpu: "4"
        memory: 8Gi
      requests:
        cpu: "2"
        memory: 4Gi
    backup_storage_class: standard-csi
    backup_storage_requirements: 7Gi
    create_backup_pvc: true
  eda:
    backup_pvc: custom-eda-backup-pvc
    backup_storage_class: standard-csi
    backup_storage_requirements: 7Gi
    create_backup_pvc: true
  hub:
    backup_pvc: custom-hub-backup-pvc
    backup_storage_class: nfs-local-rwx
    backup_storage_requirements: 7Gi
    create_backup_pvc: true

```

- a. Sets a custom PVC name for the platform gateway backup.

- b. When set to `true`, the operator creates the PVC automatically if it does not already exist.
- c. Component-level settings override the global values for `backup_pvc`, `backup_storage_class`, and `backup_storage_requirements`. Each component can define its own `backup_pvc` to create a uniquely named PVC.

2. Apply the configuration:

```
oc apply -f custom-pvc-backup.yaml
```

Result

Confirm that the PVCs were created:

```
oc get pvc -n <namespace>
```

The output displays the custom PVCs for each component:

Name	Status	Volume	Capacity	Access Modes	Storage class
custom-aap-backup-pvc	Bound	pv-aap	7Gi	RWX	nfs-local-rwx
custom-controller-backup-pvc	Bound	pv-ctrl	7Gi	RWO	standard-csi
custom-eda-backup-pvc	Bound	pv-eda	7Gi	RWO	standard-csi

Name	Status	Volume	Capacity	Access Modes	Storage class
custom-hub-backup-pvc	Bound	pv-hub	7Gi	RWX	nfs-local-rwx

Custom backup configurations for specific components

You can override the global backup configuration for specific components, such as automation controller, private automation hub, or Event-Driven Ansible controller. This enables each component to use its own storage class, PVC name, storage size, or resource limits during the backup process.

Example: Overriding component resources

In this backup custom resource, each component has a unique PVC name and storage class::

```

apiVersion: aap.ansible.com/v1alpha1
kind: AnsibleAutomationPlatformBackup
metadata:
  name: custom-backup
spec:
  deployment_name: myaap
  no_log: false
  backup_storage_class: rook-cephfs <1>
  hub:
    backup_pvc: custom-hub-backup-pvc <2>
    backup_storage_requirements: 25Gi
    backup_storage_class: rook-cephfs
    create_backup_pvc: true
    no_log: false
  controller:
    backup_pvc: custom-controller-backup-pvc
    backup_storage_requirements: 15Gi
    backup_storage_class: rook-block <3>
    create_backup_pvc: true
    no_log: false
  eda:
    backup_pvc: custom-eda-backup-pvc
    backup_storage_requirements: 7Gi
    backup_storage_class: rook-block
    create_backup_pvc: true
    no_log: false

```

1. The global `backup\_storage\_class` applies to the platform gateway. Components that do not specify their own `backup\_storage\_class` inherit this value.
2. Each component can define its own `backup\_pvc` to create a uniquely named PVC. Set `create\_backup\_pvc: true` to have the operator create the PVC automatically.
3. Component-level values, such as `backup\_storage\_class: rook-block`, override the global setting for that component.

Resolve naming conflicts when deploying an Automation Controller custom resource

If your `AutomationController` custom resource matches an existing deployment, perform the following steps to resolve the issue.

The name specified for the new `AutomationController` custom resource must not match an existing deployment or the recovery process will fail. Persistent volume claims (PVCs) and Secrets remain after a deployment is deleted. If you want to reuse the same name you must delete previous PVCs and Secrets before creating a new custom resource.

Procedure

1. Delete the existing `AutomationController` and the associated postgres PVC:

```
oc delete automationcontroller <YOUR_DEPLOYMENT_NAME> -n <YOUR_NAMESPACE>

oc delete pvc postgres-13-<YOUR_DEPLOYMENT_NAME>-13-0 -n <YOUR_NAMESPACE>
```

2. Use `AutomationControllerRestore` with the same deployment_name in it:

```
oc apply -f restore.yaml
```

Restore your Ansible Automation Platform deployment

If you lose information on your system or experience issues with an upgrade, you can use the backup resources of your deployment instances. Use the following procedures to recover your Ansible Automation Platform deployment files.

Restore Ansible Automation Platform

Ansible Automation Platform manages any enabled components (such as, automation controller, automation hub, and Event-Driven Ansible), when you recover **Ansible Automation Platform** you also restore these components.

Before you begin

- You must be authenticated with an OpenShift cluster.
- You have installed the Ansible Automation Platform Operator on the cluster.
- The **AnsibleAutomationPlatformBackups** deployment is available in your cluster.

In previous versions of the Ansible Automation Platform Operator, it was necessary to create a restore object for each component of the platform. Now, you create a single **AnsibleAutomationPlatformRestore** resource, which creates and manages the other restore objects:

- AutomationControllerRestore
- AutomationHubRestore
- EDARestore

Procedure

1. Log in to Red Hat OpenShift Container Platform.
2. Navigate to **Operators > Installed Operators**.
3. Select your Ansible Automation Platform Operator deployment.
4. Go to your **All Instances** tab, and click **Create New**.
5. Select **Ansible Automation Platform Restore** from the list.
6. For **Name** enter the name for the recovery deployment.
7. For **New Ansible Automation Platform Name** enter the new name for your Ansible Automation Platform instance.
8. **Backup Source** defaults to **CR**.
9. For **Backup name** enter the name you chose when creating the backup.
10. Click **Create**.

Result

Your backup restores under the **AnsibleAutomationPlatformRestores** tab.

NOTE:

The recovery is not complete until all the resources are successfully restored. Depending on the size of your database this can take some time.

To verify that your recovery was successful you can:

1. Go to **Workloads > Pods**.
2. Confirm that all pods are in a **Running** or **Completed** state.

Related information

[Installing on OpenShift Container Platform](#)

Restore from a persistent volume claim (PVC)

A persistent volume claim (PVC) is a storage volume that stores data for automation hub and automation controller applications.

Before you begin

- You have an existing PVC containing a backup.
- You have installed the Ansible Automation Platform Operator on Red Hat OpenShift Container Platform.

These PVCs are independent of the applications and persist even if an application is deleted. You can restore data from a PVC as an alternative to recovering from an **Ansible Automation Platform** backup.

For more information see the [Finding and deleting PVCs](#) section of the *Installing on OpenShift Container Platform* guide.

Procedure

1. Log in to Red Hat OpenShift Container Platform.
2. Navigate to **Operators > Installed Operators**.
3. Select your Ansible Automation Platform Operator deployment.
4. Go to your **All Instances** tab, and click **Create New**.
5. Select **Ansible Automation Platform Restore** from the list.
6. For **Name** enter the name for the recovery deployment.
7. For **New Ansible Automation Platform Name** enter the new name for your Ansible Automation Platform instance.
8. For **Backup Source** select **PVC**.
 - a. **Backup PVC:** Enter the name of your PVC.
 - b. **Backup Directory:** Enter the path to your backup directory on your PVC.
9. For **Backup name** enter the name you chose when creating the backup.
10. Under **YAML view** paste in the following example:

```

---
apiVersion: aap.ansible.com/v1alpha1
kind: AnsibleAutomationPlatformRestore
metadata:
  name: aap
spec:
  deployment_name: aap
  backup_source: PVC
  backup_pvc: aap-backup-claim
  backup_dir: '/backups/aap-openshift-backup-2025-06-23-18:28:29'

  controller:
    backup_source: PVC
    backup_pvc: aap-controller-backup-claim
    backup_dir: '/backups/tower-openshift-backup-2025-06-23-182910'

  hub:
    backup_source: PVC
    backup_pvc: aap-hub-backup-claim
    backup_dir: '/backups/openshift-backup-2025-06-23-182853'
    storage_type: file

  eda:
    backup_source: PVC
    backup_pvc: aap-eda-backup-claim
    backup_dir: '/backups/eda-openshift-backup-2025-06-23-18:29:11'

```

11. Click **Create**.

Result

Your backup restores under the **AnsibleAutomationPlatformRestores** tab.

NOTE:

The recovery is not complete until all the resources are successfully restored. Depending on the size of your database this can take some time.

1. Go to **Workloads > Pods**.

2. Confirm that all pods are in a **Running** or **Completed** state.

Related information

[Installing on OpenShift Container Platform](#)

Restore from an external database

You can restore an external database on Red Hat OpenShift Container Platform using the Operator. Use the following procedure to restore from an external database.

Before you begin

- You have an external database.
- You have installed the Ansible Automation Platform Operator on OpenShift Container Platform.

IMPORTANT:

Restoring from an external database force drops the database, which overrides your existing external database.

Procedure

1. Log in to Red Hat OpenShift Container Platform.
2. Navigate to **Operators > Installed Operators**.
3. Select your Ansible Automation Platform Operator deployment.
4. Go to your **All Instances** tab, and click **Create New**.
5. Select **Ansible Automation Platform Restore** from the list.
6. For **Name** enter the name for the recovery deployment.
7. For **New Ansible Automation Platform Name** enter the new name for your Ansible Automation Platform instance.
 - If restoring to the same name Ansible Automation Platform then you must add `force_drop_db: true` to drop the database on restore.
8. **Backup Source** defaults to **CR**.
9. For **Backup name** enter the name you chose when creating the backup. Under **YAML view** paste in the following example:

```

---
apiVersion: aap.ansible.com/v1alpha1
kind: AnsibleAutomationPlatformRestore
metadata:
  name: aaprestore
spec:
  deployment_name: aap
  backup_name: aapbackup
  controller:
    force_drop_db: true

```

10. Click **Create**.

Result

Your backup restores under the **AnsibleAutomationPlatformRestores** tab.

NOTE:

The recovery is not complete until all the resources are successfully restored. Depending on the size of your database this can take some time.

To verify that your recovery was successful you can:

1. Go to **Workloads > Pods**.
2. Confirm that all pods are in a **Running** or **Completed** state.

Related information

[Plan your installation of Ansible Automation Platform on Red Hat OpenShift Container Platform](#)

Back up and restore your RPM deployment

Protecting your Ansible Automation Platform RPM deployment ensures that you can recover from system failures, data corruption, or configuration errors. Regular backups minimize downtime and protect your configurations and data.

Implementing backup and restore procedures helps you to:

- **Protect critical automation assets:** Safeguard your data including configurations, inventory, credentials, and job history from unexpected failures or data loss.
- **Enable disaster recovery:** Restore your environment quickly after hardware failures or other unexpected events to maintain business continuity.

- **Support testing and development:** Create copies of your environment for testing upgrades or configuration changes without affecting production systems.

Backup and restore overview

You can backup and restore your system by using the Ansible Automation Platform setup playbook.

NOTE:

When backing up Ansible Automation Platform, use the installation program that matches your currently installed version of Ansible Automation Platform.

When restoring Ansible Automation Platform, use the latest installation program available at the time of the restore. For example, if you are restoring a backup taken from version `2.6-1`, use the latest `2.6-x` installation program available at the time of the restore.

Backup and restore functionality only works with the PostgreSQL versions supported by your current Ansible Automation Platform version. For more information, see **System requirements** in *Planning your installation*.

The Ansible Automation Platform setup playbook is invoked as `setup.sh` from the path where you unpacked the platform installer tar file. It uses the same inventory file used by the install playbook. The setup script takes the following arguments for backing up and restoring:

- `-b` : Perform a database backup rather than an installation.
- `-r` : Perform a database restore rather than an installation.

As the root user, call `setup.sh` with the appropriate parameters and the Ansible Automation Platform backup or restored as configured:

```
root@localhost:~# ./setup.sh -b
root@localhost:~# ./setup.sh -r
```

Backup files are created on the same path that `setup.sh` script exists. You can change it by specifying the following `EXTRA_VARS` :

```
root@localhost:~# ./setup.sh -e 'backup_dest=/path/to/backup_dir/' -b
```

A default restore path is used unless you provide `EXTRA_VARS` with a non-default path, as shown in the following example:

```
root@localhost:~# ./setup.sh -e 'restore_backup_file=/path/to/nondefault/
backup.tar.gz' -r
```

Optionally, you can override the inventory file used by passing it as an argument to the setup script:

```
setup.sh -i <inventory file>
```

Related information

[Backup and restore in clustered environments](#)

[Red Hat Ansible Automation Platform system requirements](#)

Backup and restore playbooks

automation controller includes playbooks to backup and restore your installation.

In addition to the `install.yml` file included with your `setup.sh` setup playbook, there are also `backup.yml` and `restore.yml` files.

These playbooks serve to backup and restore.

- The overall backup, backs up:
 - The database
 - The `SECRET_KEY` file
- The per-system backups include:
 - Custom configuration files
 - Manual projects
- The restore backup restores the backed up files and data to a freshly installed and working second instance of automation controller.

When restoring your system, installation program checks to see that the backup file exists before beginning the restoration. If the backup file is not available, your restoration fails.

NOTE:

Make sure that your automation controller hosts are properly set up with SSH keys, user or pass variables in the hosts file, and that the user has `sudo` access.

Considerations for backup and restore

Consider the following points when you backup and restore your system:

Disk space

Review your disk space requirements to ensure you have enough room to backup configuration files, keys, other relevant files, and the database of the Ansible Automation Platform installation.

System credentials

Confirm you have the required system credentials when working with a local database or a remote database. On local systems, you might need `root` or `sudo` access, depending on how credentials are set up. On remote systems, you might need different credentials to grant you access to the remote system you are trying to backup or restore.

NOTE:

The Ansible Automation Platform database backups are staged on each node at `/var/backups/automation-platform` through the variable `backup_dir`. You might need to mount a new volume to `/var/backups` or change the staging location with the variable `backup_dir` to prevent issues with disk space before running the `./setup.sh -b` script.

Version

You must always use the most recent minor version of a release to backup or restore your Ansible Automation Platform installation version. For example, if the current platform version you are on is 2.0.x, only use the latest 2.0 installer.

File path

When using `setup.sh` to do a restore from the default restore file path, `/var/lib/awx`, `-r` is still required to do the restore, but it no longer accepts an argument. If a non-default restore file path is needed, you must provide this as an extra_var (`root@localhost:~# ./setup.sh -e 'restore_backup_file=/path/to/nondefault/backup.tar.gz' -r`).

Directory

If the backup file is placed in the same directory as the `setup.sh` installer, the restore playbook automatically locates the restore files. In this case, you do not need to use the `restore_backup_file` extra var to specify the location of the backup file.

Back up your Ansible Automation Platform instance

Back up an Ansible Automation Platform instance by running the `setup.sh` script with the `backup_dest` flag. You can also enable the compression flags `use_archive_compression` and `use_db_compression` to reduce the size of the backup artifacts.

Procedure

1. Navigate to your Ansible Automation Platform installation directory.
2. Run the `./setup.sh` script following the example below:

```
$ ./setup.sh -e 'backup_dest=/ansible/mybackup' -e
'use_archive_compression=true' 'use_db_compression=true @credentials.yml -b
```

Where:

- `backup_dest` : Specifies a directory to save your backup to.
- `use_archive_compression=true` and `use_db_compression=true` : Compresses the backup artifacts before they are sent to the host running the backup operation. You can use the following variables to customize the compression:
 - For global control of compression for filesystem related backup files: `use_archive_compression=true`
 - For component-level control of compression for filesystem related backup files: `<componentName>_use_archive_compression`
For example:
 - `automationgateway_use_archive_compression=true`
 - `automationcontroller_use_archive_compression=true`
 - `automationhub_use_archive_compression=true`
 - `automationedacontroller_use_archive_compression=true`
 - For global control of compression for database related backup files: `use_db_compression=true`
 - For component-level control of compression for database related backup files: `<componentName>_use_db_compression=true`
For example:
 - `automationgateway_use_db_compression=true`
 - `automationcontroller_use_db_compression=true`

- `automationhub_use_db_compression=true`
- `automationedacontroller_use_db_compression=true`

Result

After a successful backup, a backup file is created at `/ansible/mybackup/automation-platform-backup-
backup-
<date/time>.tar.gz`.

Backup and restore in clustered environments

Learn how to back up and restore automation controller in a clustered environment.

The procedure for backup and restore for a clustered environment is similar to a single install, except for some of the following considerations:

- If restoring to a new cluster, ensure that the old cluster is shut down before proceeding because they can conflict with each other when accessing the database.
- Per-node backups are only restored to nodes bearing the same hostname as the backup.
- When restoring to an existing cluster, the restore has the following:
 - A dump of the PostgreSQL database
 - UI artifacts, included in the database dump
 - An automation controller configuration (retrieved from `/etc/tower`)
 - An automation controller secret key
 - Manual projects

Restore to a different cluster

When restoring a backup to a separate instance or cluster, manual projects and custom settings under `/etc/tower` are retained. Job output and job events are stored in the database, and therefore, not affected.

The restore process does not alter instance groups present before the restore. It does not introduce any new instance groups either. Restored automation controller resources that were associated to instance groups likely need to be reassigned to instance groups present on the new automation controller cluster.

WARNING:

If you plan to use a backup and restore operation to migrate your Ansible Automation Platform instance to a different cluster or new set of hostnames, any Red Hat Ansible Automation Platform credentials you set up for Event-Driven Ansible controller will break, causing your rulebook activations to fail. You must manually edit and update the automation controller URL and associated credentials after the restore operation is complete to restore connectivity.

Define, create, and build execution environments

Run automation consistently across nodes with execution environments, which are container images that contain everything you need to run your automation.

An execution environment is a containerized runtime that provides the following benefits:

- A consistent environment in which to run automation jobs
- Portability and scalability, as you can run automation jobs on any node, including controller and execution nodes
- Security and governance, as you can control what's inside the runtime environment; and you can have approved, signed, and verified container images
- Improved efficiency, as developers need not spend time troubleshooting environment dependencies

Execution environments contain:

- Ansible core
- Required collections
- Python and other system dependencies
- Other libraries that your automation may require

Ansible Builder is a command line tool that automates the process of building automation execution environments by using metadata defined in various Ansible Collections or created by the user. You build an execution environment before you can create it using automation controller. After building it, you push it to a repository (such as quay) and then, when creating an execution environment in the UI with automation controller, you must point to that repository to use it in Ansible Automation Platform to use it, for example, in a job template.

With Ansible Builder, you can easily create a customizable automation execution environments definition file that specifies the content you want included in your automation execution environments such as Ansible Core, Python, Collections, third-party Python requirements, and system level packages. This enables you to fulfill all of the necessary requirements and dependencies to get jobs running.

Build a definition file

You can use Ansible Builder to create an execution environment. Building a new execution environment involves a definition that specifies which content you want to include in your execution environment, such as collections, Python requirements, and system-level packages.

After you install Ansible Builder, you can create a definition file that Ansible Builder uses to create your automation execution environment image. Ansible Builder makes an automation execution environment image by reading and validating your definition file, then creating a `Containerfile`, and finally passing the `Containerfile` to Podman, which then packages and creates your automation execution environment image. The definition file that you create must be in `YAML` format, with a `.yaml` or `.yml` extension, and contain different sections. The default definition filename, if not provided, is `execution-environment.yml`. For more information on the parts of a definition file, see *Breakdown of definition file content*.

The following is an example of a version 3 definition file. Each definition file must specify the major version number of the Ansible Builder feature set it uses. If not specified, Ansible Builder defaults to version 1, making most new features and definition keywords unavailable.

```
version: 3

build_arg_defaults:
  ANSIBLE_GALAXY_CLI_COLLECTION_OPTS: '--pre'

dependencies:
  galaxy: requirements.yml
  python:
    - six
    - psutil
  system: bindep.txt

images:
  base_image:
    name: registry.redhat.io/ansible-automation-platform-26/ee-minimal-rhel9:latest

# Custom package manager path for the RHEL based images
options:
  package_manager_path: /usr/bin/microdnf

additional_build_steps:
```

```

prepend_base:
  - RUN echo This is a prepend base command!

prepend_galaxy:
  # Environment variables used for Galaxy client configurations
  - ENV ANSIBLE_GALAXY_SERVER_LIST=automation_hub
  - ENV ANSIBLE_GALAXY_SERVER_AUTOMATION_HUB_URL=https://console.redhat.com/api/automation-hub/content/published/
  - ENV ANSIBLE_GALAXY_SERVER_AUTOMATION_HUB_AUTH_URL=https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token
  # define a custom build arg env passthru - we still also have to pass
  # `--build-arg ANSIBLE_GALAXY_SERVER_AUTOMATION_HUB_TOKEN` to get it to pick it up from the env
  - ARG ANSIBLE_GALAXY_SERVER_AUTOMATION_HUB_TOKEN

prepend_final: |
  RUN whoami
  RUN cat /etc/os-release
append_final:
  - RUN echo This is a post-install command!
  - RUN ls -la /etc

```

- `build_arg_defaults` : Lists default values for build arguments.
- `dependencies` : Specifies the location of various requirements files.
- `images` : Specifies the base image to be used. Red Hat support is only provided for the `redhat.registry.io` base image.
- `options` : Specifies options that can affect builder runtime functionality.
- `additional_build_steps` : Commands for additional custom build steps.

Related information

[Definition file reference](#)

[Ansible Builder Porting Guide](#)

Definition dependencies

You can include dependencies that must be installed into the final image in the `dependencies` section of your definition file.

To avoid issues with your automation execution environment image, make sure that the entries for Galaxy, Python, and system point to a valid requirements file, or are valid content for their file types.

Galaxy

The `galaxy` entry points to a valid requirements file or includes inline content for the `ansible-galaxy collection install -r ...` command.

The entry `requirements.yml` can be a relative path from the directory of the automation execution environment definition's folder, or an absolute path.

The content might look like the following:

```
collections:
  - community.aws
  - kubernetes.core
```

Python

The `python` entry in the definition file points to a valid requirements file or to an inline list of Python requirements in PEP508 format for the `pip install -r ...` command.

The entry `requirements.txt` is a file that installs extra Python requirements on top of what the Collections already list as their Python dependencies. It might be listed as a relative path from the directory of the automation execution environment definition's folder, or an absolute path. The contents of a `requirements.txt` file should be formatted as the following example, similar to the standard output from a `pip freeze` command.

The content might look like the following:

```

boto>=2.49.0
botocore>=1.12.249
pytz
python-dateutil>=2.7.0
awxkit
packaging
requests>=2.4.2
xmldict
azure-cli-core==2.11.1
openshift>=0.6.2
requests-oauthlib
openstacksdk>=0.13
ovirt-engine-sdk-python>=4.4.10

```

Related information

[PEP508 format standard](#)

System

The `system` entry in the definition points to a `bindep` requirements file or to an inline list of `bindep` entries, which install system-level dependencies that are outside of what the collections already include as their dependencies.

The `system` entry can be listed as a relative path from the directory of the automation execution environment definition's folder, or as an absolute path. At a minimum, the collections must specify necessary requirements for `[platform:rpm]`.

To demonstrate this, the following is an example `bindep.txt` file that adds the `libxml2` and `subversion` packages to a container.

The content might look like the following:

```

libxml2-devel [platform:rpm]
subversion [platform:rpm]

```

Entries from multiple collections are combined into a single file. This is processed by `bindep` and then passed to `dnf`. Only requirements with no profiles or no runtime requirements will be installed to the image.

Related information

[bindep documentation](#)

Images

The `images` section of the definition file identifies the base image. Verification of signed container images is supported with the `podman` container runtime.

The following table shows a list of values that you can use in `images`:

Value	Description
<code>base_image</code>	Specifies the parent image for the automation execution environment which enables a new image to be built that is based on an existing image. This is typically a supported execution environment base image such as <i>ee-minimal</i> or <i>ee-supported</i>, but it can also be an execution environment image that you have created and want to customize further. A <code>name</code> key is required for the container image to use. Specify the <code>signature</code> <code>_original_name</code> key if the image is mirrored within your repository, but is signed with the image's original signature key. Image names must contain a tag, such as <code>:latest</code>. The default image is <code>registry.redhat.io/ansible-automation-platform-24/ee-minimal-rhel8:latest</code>.

Disconnected environment customizations

Creating execution environments for Ansible Automation Platform is a common task which works differently in disconnected environments. When building a custom execution environment, the `ansible-builder` tool defaults to downloading content from the following locations on the internet:

- Red Hat Automation hub (console.redhat.com) or Ansible Galaxy (galaxy.ansible.com) for any Ansible content collections added to the execution environment image.
- PyPI (pypi.org) for any python packages required as collection dependencies.
- RPM repositories such as the RHEL or UBI repositories (cdn.redhat.com) for adding or updating RPMs to the execution environment image, if needed.
- `registry.redhat.io` for access to the base container images.

Building an execution environment image in a disconnected environment requires mirroring content from these locations. For information about importing collections from Ansible Galaxy or automation hub into a private automation hub, see *Importing an automation content collection in automation hub* in the Related Links section.

Mirrored PyPI content once transferred into the disconnected network can be made available by using a web server or an artifact repository such as Nexus. The RHEL and UBI repository content can be exported from an internet-facing Red Hat Satellite Server, copied into the disconnected environment, then imported into a disconnected Satellite so it is available for building custom execution environments. See *ISS Export Sync in an Air-Gapped Scenario* in the Related Links section for details.

The default base container image, `ee-minimal-rhel8`, is used to create custom execution environment images and is included with the bundled installer. This image is added to the private automation hub at install time.

If a different base container image such as `ee-minimal-rhel9` is required, it must be imported to the disconnected network and added to the private automation hub container registry.

Once all of the prerequisites are available on the disconnected network, the `ansible-builder` command can be used to create custom execution environment images.

Related information

[Importing an automation content collection in automation hub](#)
[ISS export sync in an air-gapped scenario](#)

Execution environment setup reference

Review this reference information to better understand the definition of an execution environment.

You define the content of your execution environment in a YAML file. By default, this file is called `execution_environment.yml`. This file tells Ansible Builder how to create the build instruction file (Containerfile for Podman, Dockerfile for Docker) and build context for your container image.

NOTE:

The definition schema for Ansible Builder 3.x is documented here. If you are running an older version of Ansible Builder, you need an older schema version. We recommend using version 3, which offers substantially more configurable options and functionality than previous versions.

Related information

[Introduction to Ansible Builder](#)

Execution environment definition example

You must create a definition file to build an image for an execution environment. The file is in YAML format.

You must specify the version of Ansible Builder in the definition file. The default version is 1.

The following definition file is using Ansible Builder version 3:

```
version: 3
build_arg_defaults:
  ANSIBLE_GALAXY_CLI_COLLECTION_OPTS: '--pre'
dependencies:
  galaxy: requirements.yml
  python:
    - six
    - psutil
  system: bindep.txt
images:
  base_image:
    name: registry.redhat.io/ansible-automation-platform-24/ee-minimal-rhel8:latest
additional_build_files:
  - src: files/ansible.cfg
    dest: configs
additional_build_steps:
  prepend_galaxy:
    - ADD _build/configs/ansible.cfg /home/runner/.ansible.cfg
  prepend_final: |
    RUN whoami
    RUN cat /etc/os-release
  append_final:
    - RUN echo This is a post-install command!
    - RUN ls -la /etc
```

Configuration options

When defining a custom execution environment for use with automation controller, you can specify various configuration options in the definition file used by Ansible Builder to build the EE image.

Use the following configuration YAML keys in your definition file.

The Ansible Builder 3.x execution environment definition file accepts seven top-level sections:

- [Additional_build_files](#)

- [Additional_build_steps](#)
- [build_arg_defaults](#)
- [Dependencies](#)
- [Images](#)
 - [Image verification](#)
- [Options](#)
- [Version](#)

Additional_build_files

Various build steps can require additional files to be present in the build context directory. These files can be specified using the `additional_build_files` configuration item in the `controller_settings` section of the `controller_configuration.yml` file.

The build files specify what are to be added to the build context directory. These can then be referenced or copied by `additional_build_steps` during any build stage.

The format is a list of dictionary values, each with a `src` and `dest` key and value.

Each list item must be a dictionary containing the following required keys:

src	Specifies the source files to copy into the build context directory. This can be an absolute path, for example, <code>/home/user/.ansible.cfg</code>, or a path that is relative to the file. Relative paths can be a glob expression matching one or more files, for example, <code>files/*.cfg</code>. Note that an absolute path must not include a regular expression. If <code>src</code> is a directory, the entire contents of that directory are copied to <code>dest</code>.
dest	Specifies a subdirectory path underneath the <code>_build</code> subdirectory of the build context directory that contains the source files, for example, <code>files/configs</code>. This cannot be an absolute path or contain <code>..</code> within the path. This directory is created for you if it does not exist. NOTE: When using an <code>ansible.cfg</code> file to pass a token and other settings for a private account to an automation hub server, listing the configuration file path here as a string enables it to be included as a build argument in the initial phase of the build.

Additional_build_steps

The build steps specify custom build commands for any build phase. These commands are inserted directly into the build instruction file for the container runtime, for example, Containerfile or Dockerfile. The commands must conform to any rules required by the containerization tool.

You can add build steps before or after any stage of the image creation process. For example, if you need `git` to be installed before you install your dependencies, you can add a build step at the end of the base build stage.

The following are the valid keys. Each supports either a multi-line string, or a list of strings.

append_base	Commands to insert after building of the base image.
append_builder	Commands to insert after building of the builder image.
append_final	Commands to insert after building of the final image.
append_galaxy	Commands to insert after building of the galaxy image.
prepend_base	Commands to insert before building of the base image.
prepend_builder	Commands to insert before building of the builder image.
prepend_final	Commands to insert before building of the final image.
prepend_galaxy	Commands to insert before building of the galaxy image.

build_arg_defaults

This specifies the default values for build arguments as a dictionary.

This is an alternative to using the `--build-arg` CLI flag.

Ansible Builder uses the following build arguments:

ANSIBLE_GALAXY_CLI_COLLECTION_OPTS	Enables the user to pass the <code>-pre</code> flag and other flags to enable the installation of pre-release collections.
ANSIBLE_GALAXY_CLI_ROLE_OPTS	This enables the user to pass any flags, such as <code>--no-deps</code> , to the role installation.

PKGMR_PRESERVE_CACHE

This controls how often the package manager cache is cleared during the image build process.

If this value is not set, which is the default, the cache is cleared frequently. If the value is `always`, the cache is never cleared. Any other value forces the cache to be cleared only after the system dependencies are installed in the final build stage.

Ansible Builder hard-codes values given inside of `build_arg_defaults` into the build instruction file, so they persist if you run your container build manually.

If you specify the same variable in the definition and at the command line with the CLI `build-arg` flag, the CLI value overrides the value in the definition.

Dependencies

Specifies dependencies to install into the final image, including `ansible-core`, `ansible-runner`, Python packages, system packages, and collections.

Ansible Builder automatically installs dependencies for any Ansible collections you install.

In general, you can use standard syntax to constrain package versions. Use the same syntax you would pass to `dnf`, `pip`, `ansible-galaxy`, or any other package management utility. You can also define your packages or collections in separate files and reference those files in the `dependencies` section of your definition file.

The following keys are valid:

ansible_core

The version of the `ansible-core` Python package to be installed.

This value is a dictionary with a single key, `package_pip`. The `package_pip` value is passed directly to `pip` for installation and can be in any format that `pip` supports. The following are some example values:

	<pre> ansible_core: package_pip: ansible-core ansible_core: package_pip: ansible-core==2.14.3 ansible_core: package_pip: https://github.com/example_user/ ansible/archive/refs/heads/ansible.tar.gz </pre>
ansible_runner	The version of the Ansible Runner Python package to be installed. This value is a dictionary with a single key, <code>package_pip</code>. The <code>package_pip</code> value is passed directly to pip for installation and can be in any format that pip supports. The following are some example values: <pre> ansible_runner: package_pip: ansible-runner ansible_runner: package_pip: ansible-runner==2.3.2 ansible_runner: package_pip: https://github.com/example_user/ ansible-runner/archive/refs/heads/ansible- runner.tar.gz </pre>
galaxy	Collections to be installed from Ansible Galaxy. This can be a filename, a dictionary, or a multi-line string representation of an Ansible Galaxy <code>requirements.yml</code> file.
python	The Python installation requirements. This can be a filename, or a list of requirements. Ansible Builder combines all the Python requirements files from all collections into a single file by using the <code>requirements-parser</code> library. This library supports complex syntax, including references to other files. If many collections require the same <i>package name</i>, Ansible Builder combines them into a single entry and combines the constraints.

	Ansible Builder excludes some packages in the combined file of Python dependencies even if a collection lists them as dependencies. These include test packages and packages that provide Ansible itself. The full list can be available under <code>EXCLUDE_REQUIREMENTS</code> in <code>src/ansible_builder/_target_scripts/introspect.py</code>. If you need to include one of these excluded package names, use the <code>--user-pip</code> option of the <code>introspect</code> command to list it in the user requirements file. Packages supplied this way are not processed against the list of excluded Python packages.
python_interpreter	A dictionary that defines the Python system package name to be installed by dnf (<code>package_system</code>) or a path to the Python interpreter to be used (<code>python_path</code>) .
system	The system packages to be installed, in bindep format. This can be a filename or a list of requirements. For more information about bindep, see the OpenDev documentation. For system packages, use the <code>bindep</code> format to specify cross-platform requirements, so they can be installed by whichever package management system the execution environment uses. Collections must specify necessary requirements for <code>[platform:rpm]</code> . Ansible Builder combines system package entries from multiple collections into a single file. Only requirements with no profiles (runtime requirements) are installed to the image. Entries from many collections that are duplicates of each other can be consolidated in the combined file.

The following example uses filenames that contain the various dependencies:

```
dependencies:
  python: requirements.txt
  system: bindep.txt
  galaxy: requirements.yml
  ansible_core:
    package_pip: ansible-core==2.14.2
  ansible_runner:
    package_pip: ansible-runner==2.3.1
  python_interpreter:
    package_system: "python310"
    python_path: "/usr/bin/python3.10"
```

This example uses inline values:

```
dependencies:
  python:
    - pywinrm
  system:
    - iputils [platform:rpm]
  galaxy:
    collections:
      - name: community.windows
      - name: ansible.utils
      version: 2.10.1
  ansible_core:
    package_pip: ansible-core==2.14.2
  ansible_runner:
    package_pip: ansible-runner==2.3.1
  python_interpreter:
    package_system: "python310"
    python_path: "/usr/bin/python3.10"
```

NOTE:

If any of these dependency files (`requirements.txt`, `bindep.txt`, and `requirements.yml`) are in the `build_ignore` of the collection, the build fails.

Collection maintainers can verify that ansible-builder recognizes the requirements they expect by using the `introspect` command:

```
ansible-builder introspect --sanitize ~/.ansible/collections/
```

The `--sanitize` option reviews all of the collection requirements and removes duplicates. It also removes any Python requirements that are normally excluded (see **python** dependencies).

Use the `-v3` option to `introspect` to see logging messages about requirements that are being excluded.

Images

Use the **images** dictionary to define container images for your execution environment. Each key represents a unique image name, while the corresponding value is a dictionary defining that image's properties.

At a minimum you must specify a source, image, and tag for the base image. The base image provides the operating system and can also provide some packages. Use the standard `host/namespace/container:tag` syntax to specify images. You can use Podman or Docker shortcut syntax instead, but the full definition is more reliable and portable.

Valid keys for this section are:

base_image
A dictionary defining the parent image for the execution environment. A <code>name</code> key must be supplied with the container image to use. Use the <code>signature_original_name</code> key if the image is mirrored within your repository, but signed with the original image's signature key.

Image verification

You can verify signed container images if you are using the `podman` container runtime.

Set the `container-policy` CLI option to control how this data is used in relation to a Podman `policy.json` file for container image signature validation.

- **ignore_all** policy: Generate a `policy.json` file in the build `context` directory `<context>` where no signature validation is performed.
- **system** policy: Signature validation is performed using pre-existing `policy.json` files in standard system locations. `ansible-builder` assumes no responsibility for the content within these files, and the user has complete control over the content.
- **signature_required** policy: `ansible-builder` uses the container image definitions to generate a `policy.json` file in the build `context` directory `<context>` that is used during the build to validate the images.

Options

A dictionary of keywords or options that can affect the runtime functionality Ansible Builder.

Valid keys for this section are:

- **container_init**: A dictionary with keys that allow for customization of the container `ENTRYPOINT` and `CMD` directives (and related behaviors). Customizing these behaviors is an advanced task, and can result failures that are difficult to debug. Because the provided defaults control several intertwined behaviors, overriding any value skips all remaining defaults in this dictionary.
Valid keys are:
 - **cmd**: Literal value for the `CMD` Containerfile directive. The default value is `["bash"]`.
 - **entrypoint**: Literal value for the `ENTRYPOINT` Containerfile directive. The default entrypoint behavior handles signal propagation to subprocesses, and attempting to ensure at runtime that the container user has a proper environment with a valid writeable home directory, represented in `/etc/passwd`, with the `HOME` environment variable set to match. The default entrypoint script can emit warnings to `stderr` in cases where it is unable to suitably adjust the user runtime environment. This behavior can be ignored or elevated to an unrecoverable error; consult the source for the `entrypoint` target script for more details.
 - **package_pip**: Package to install with pip for entrypoint support. This package is installed in the final build image.
- **package_manager_path**: string with the path to the package manager (dnf or microdnf) to use. This value is used to install packages.
- **skip_ansible_check**: This boolean value controls whether or not the check for an installation of Ansible and Ansible Runner is performed on the final image. Set this value to `True` to not perform this check.

The default is `False`.

- **relax_passwd_permissions**: This boolean value controls whether the `root` group (GID 0) is explicitly granted write permission to `/etc/passwd` in the final container image. The default entrypoint script can attempt to update `/etc/passwd` under some container

runtimes with dynamically created users to ensure a fully-functional POSIX user environment and home directory. Disabling this capability can cause failures of software features that require users to be listed in `/etc/passwd` with a valid and writeable home directory, for example, `async` in `ansible-core`, and the `~username` shell expansion. The default is `True`.

- **workdir:** Default current working directory for new processes started under the final container image. Some container runtimes also use this value as `HOME` for dynamically-created users in the `root` (GID 0) group. When this value is specified, if the directory does not already exist, it is created, set to `root` group ownership, and `rxwx` group permissions are recursively applied to it. The default value is `/runner`.
- **user:** This sets the username or UID to use as the default user for the final container image. The default value is `1000`.

Example

```
options:
  container_init:
    package_pip: dumb-init>=1.2.5
    entrypoint: ["dumb-init"]
    cmd: ["csh"]
  package_manager_path: /usr/bin/microdnf
  relax_password_permissions: false
  skip_ansi_check: true
  workdir: /myworkdir
  user: bob
```

Version

An integer value that sets the schema version of the execution environment definition file.

Defaults to `1`.

The value must be `3` if you are using Ansible Builder 3.x.

Set up your container repository

When you set up your container repository, you can add a description, include a README, add teams that can access the repository, and tag automation execution environments.

Your container repository for Private Automation Hub is where execution environments are stored, signed, and retrieved by automation controllers.

Add a README to your container repository

Add a README to your container repository to provide instructions to your users on how to work with the container. Automation hub container repositories support Markdown for creating a README. By default, the README is empty.

Before you begin

- You have permissions to change containers.

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Automation Content** > **Execution Environments**.
3. Select your execution environment.
4. On the **Detail** tab, click **Add**.
5. In the **Raw Markdown** text field, enter your README text in Markdown.
6. Click **Save** when you are finished.

Next steps

After you add a README, you can edit it at any time by clicking **Edit** and repeating steps 4 and 5.

Provide access to your automation execution environments

Provide users with access to your automation execution environments by adding them to a team. Then, modify the permissions the team can have to a particular execution environment.

Before you begin

- You have **change container namespace** permissions.

Procedure

1. Log in to Ansible Automation Platform.

2. From the navigation panel, select **Automation Content > Execution Environments**.
3. Select your automation execution environment.
4. From the **Team Access** tab, click **Add roles**.
5. Select the team or teams to which you want to grant access and click **Next**.
6. Select the roles that you want to add to this execution environment and click **Next**.
7. Click **Finish**.

Tag container images

Tag automation execution environments to add an additional name to automation execution environments stored in your automation hub container repository. If no tag is added to an automation execution environment, automation hub defaults to `latest` for the name.

Before you begin

- You have **change automation execution environment tags** permissions.

Procedure

1. From the navigation panel, select **Automation Content > Execution Environments**.
2. Select your automation execution environments.
3. Click the **Images** tab.
4. Click the **More Actions** icon , and click **Manage tags**.
5. Add a new tag in the text field and click **Add**.
6. Optional: Remove **current tags** by clicking **x** on any of the tags for that image.

Result

- Click the **Activity** tab and review the latest changes.

Create a credential

To pull automation execution environments images from a password or token-protected registry, you must create a credential.

In earlier versions of Ansible Automation Platform, you were required to deploy a registry to store execution environment images. On Ansible Automation Platform 2.0 and later, the system operates as if you already have a remote registry up and running. To store execution environment images, add the credentials of only your selected remote registries.

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Automation Execution > Infrastructure > Credentials**.
3. Click **Create credential** to create a new credential.
4. Enter an authorization **Name**, **Description**, and **Organization**.
5. In the **Credential Type** drop-down, select **Container Registry**.
6. Enter the **Authentication URL**. This is the remote registry address.
7. Enter the **Username** and **Password or Token** required to log in to the remote registry.
8. Optional: To enable SSL verification, select **Verify SSL**.
9. Click **Create credential**.

Pull execution environments for use in automation hub

Before you can push execution environments to your private automation hub, you must first pull them from an existing registry and tag them for use.

Before you begin

- You have permissions to pull automation execution environments from `registry.redhat.io`.

The following example details how to pull an execution environment from the Red Hat Ecosystem Catalog (`registry.redhat.io`).

Procedure

1. Log in to Podman with your `registry.redhat.io` credentials:

```
$ podman login registry.redhat.io
```

2. Pull an execution environment:

```
$ podman pull registry.redhat.io/<ee_name>:<tag>
```

Result

To verify that the execution environment you pulled is contained in the list, take these steps:

1. List the images in local storage:

```
$ podman images
```

2. Check the execution environment name, and verify that the tag is correct.

Related information

[Red Hat Ecosystem Catalog Help](#)

Tag execution environments

After you pull execution environments from a registry, tag them for use in your private automation hub remote registry.

Before you begin

- You have pulled an execution environment from an external registry.
- You have the FQDN or IP address of the automation hub instance.
- Tag a local execution environment with the automation hub container repository:

```
$ podman tag registry.redhat.io/<ee_name>:<tag> <automation_hub_hostname>/<ee_name>
```

Result

1. List the images in local storage:

```
$ podman images
```

2. Verify that the execution environment you tagged with your automation hub information is contained in the list.

Push an execution environment to private automation hub

You can push tagged execution environments to private automation hub to create new containers and populate the remote registry.

Before you begin

- You have permissions to create new containers.
- You have the FQDN or IP address of the Ansible Automation Platform instance.

Procedure

1. Log in to Podman using your Ansible Automation Platform location and credentials:

```
$ podman login -u=<username> -p=<password> <aap_url>
```

WARNING:

Let Podman prompt you for your password when you log in. Entering your password at the same time as your username can expose your password to the shell history.

2. Push your execution environment to your automation hub remote registry:

```
$ podman push <automation_hub_url>/<ee_name>
```

Result

1. Log in to Ansible Automation Platform.
2. Navigate to **Automation Content > Execution Environments**.
3. Locate the container in the container repository list.

The `push` operation re-compresses image layers during the upload, which is not guaranteed to be reproducible and is client-implementation dependent. This may lead to image-layer digest changes and a failed push operation, resulting in `Error: Copying this image requires changing layer representation, which is not possible (image is signed or the destination specifies a digest)`.

Pull and sync images from automation hub to your local system

Pull Ansible Automation Platform execution environments from the automation hub registry to your local machine. Use the provided `podman pull` command for the `latest` version in the repository, or specify a tag to copy a specific execution environment.

Pull an image

Use the user interface to pull an execution environment from your private automation hub remote registry to make a copy to your local machine.

Before you begin

- You must have permission to view and pull from a private container repository.
- If you are pulling automation execution environments from a password or token-protected registry, [create a credential](#) first.

Procedure

1. From the navigation panel, select **Automation Content > Execution Environments**.
2. Select your execution environment.
3. In the **Pull this image** entry, click **Copy to clipboard**.
4. Paste and run the command in your terminal.

Result

- Run `podman images` to view images on your local machine.

Sync images from a container registry

You can pull automation execution environments from the private automation hub remote registry to sync an image to your local machine. To sync an execution environment from a remote registry, you must first configure a remote registry.

Before you begin

You must have permission to view and pull from a private container repository.

Procedure

1. From the navigation panel, select **Automation Content > Execution Environments**.
2. Add <https://registry.redhat.io> to the registry.
3. Add any required credentials to authenticate.

NOTE:

Some remote registries are aggressive with rate limiting. Set a rate limit under **Advanced Options**.

4. From the navigation panel, select **Automation Content > Execution Environments**.
5. Click **Create execution environment** in the page header.

6. Select the registry you want to pull from. The **Name** field displays the name of the automation execution environments displayed on your local registry.

NOTE:

The **Upstream name** field is the name of the image on the remote server. For example, if the upstream name is set to "alpine" and the **Name** field is "local/alpine", the alpine image is downloaded from the remote and renamed to "local/alpine".

7. Set a list of tags to include or exclude. Note that syncing automation execution environments with a large number of tags is time consuming and uses a lot of disk space.

Related information

[Red Hat Container Registry Authentication](#)

[What is Podman?](#)

Secure your automation with container signing

Add an extra layer of security to your content by enabling container signing in private automation hub.

Deploy your system for container signing

For added security, set up your system for container signing.

Before you begin

Automation content collection and container signing must be enabled.

Note that installer looks for the script and key on the same server where installer is located.

Procedure

1. From a terminal, create a signing script, and pass the script path as an installer parameter.

Example:

```
#!/usr/bin/env bash

# pulp_container SigningService will pass the next 4 variables to the script.
MANIFEST_PATH=$1
FINGERPRINT="$PULP_SIGNING_KEY_FINGERPRINT"
IMAGE_REFERENCE="$REFERENCE"
SIGNATURE_PATH="$SIG_PATH"

# Create container signature using skopeo
skopeo standalone-sign \
    $MANIFEST_PATH \
    $IMAGE_REFERENCE \
    $FINGERPRINT \
    --output $SIGNATURE_PATH

# Optionally pass the passphrase to the key if password protected.
# --passphrase-file /path/to/key_password.txt

# Check the exit status
STATUS=$?
if [ $STATUS -eq 0 ]; then
    echo {"signature_path": "$SIGNATURE_PATH"}
else
    exit $STATUS
fi
```

2. Review the Ansible Automation Platform installer inventory file for options for container signing that begin with `automationhub_*`.

```
[all:vars]
.
.
.

automationhub_create_default_container_signing_service = True
automationhub_container_signing_service_key = /absolute/path/to/key/to/sign
automationhub_container_signing_service_script = /absolute/path/to/script/
that/signs
```

3. Once installation is complete, log in to Ansible Automation Platform and navigate to **Automation Content > Signature Keys**.

NOTE:

The `container-default` service is created by the Ansible Automation Platform installer.

Result

Ensure that you have a key titled **container-default**, or **container-*anyname***.

Related information

[Enable automation content collection and container signing](#)

[Add and sign an execution environment](#)

Add and sign an execution environment

Push a signed execution environment to your private automation hub for added security.

Procedure

1. From the navigation panel, select **Automation Content > Execution Environments**.
2. Click **Create execution environment** and enter the relevant information in the fields that appear.
 - a. The **Name** field displays the name of the execution environment on your local registry.
 - b. The **Upstream name** field is the name of the image on the remote server.
 - c. Under **Registry**, select the name of the registry from the drop-down menu.
 - d. Optional: Enter tags in the **Add tag(s) to include** field. If the field is blank, all the tags are passed. You must specify which repository-specific tags to pass.

- e. Optional: Enter tags to exclude in the **Add tag(s) to exclude** field.
3. Click **Create execution environment**. You should see your new execution environment in the list that appears.
4. Sync and sign your new automation execution environment.
 - a. Click the **More Actions** icon  and select **Sync execution environment**.
 - b. Click the **More Actions** icon  and select **Sign execution environment**.
5. Click on your new execution environment. On the Details page, find the **Signed** label to determine that your execution environment has been signed.

Push a signed execution environment from your local system

Sign an automation execution environment on a local system and push the signed execution environment to the automation hub registry.

Procedure

1. From a terminal, log in to Podman, or any container client currently in use, and pull the execution environment you want to sign.

```
podman pull <container-name>
```

2. After the execution environment is pulled, add tags (for example: `latest`, `rc`, `beta`, or version numbers, such as `1.0`, `2.3`, and so on):

```
podman tag <container-name> <server-address>/<container-name>:<tag name>
```

3. Sign the execution environment after changes have been made, and push it back up to the automation hub registry:

```
podman push <server-address>/<container-name>:<tag name> --tls-verify=false  
--sign-by <reference to the gpg key on your local>
```

If the execution environment is not signed, it can only be pushed with any current signature embedded. Alternatively, you can use the following script to push the execution environment without signing it:

```
podman push <server-address>/<container-name>:<tag name> --tls-verify=false
```

4. After the execution environment has been pushed, navigate to **Automation Content > Execution Environments**.
5. To display the new execution environment, click the **Refresh** icon.
6. Click the name of the image to view your pushed image.

Result

After the execution environment is signed, the status changes to "signed".

The details page for each execution environment indicates whether it has been signed. If the details page indicates that an image is **Unsigned**, you can sign the execution environment from automation hub using the following steps:

1. Click the execution environment name to navigate to the details page.
2. Click the **More Actions** icon . Three options are available:
 - **Sign execution environment**
 - **Use in Controller**
 - **Delete execution environment**
3. Click **Sign execution environment** from the drop-down menu.

Confirm signatures are on your local environment

Podman and other image clients can use policies to ensure the validity of an image. To enable this capability, assign a policy to the signature.

To ensure an execution environment is signed by specific signatures, the signatures must first be on your local environment.

Procedure

1. From the navigation panel, select **Automation Content > Signature Keys**.
2. Click the **Download key** icon next to the signature that you are using. A new window should open to indicate you have downloaded the key.

Configure a client to verify signatures

To ensure an execution environment pulled from the remote registry is properly signed, first configure the execution environment with the proper public key in a policy file.

Before you begin

- The client must have sudo privileges configured to verify signatures.

Procedure

1. Open your terminal and use the following command:

```
sudo <name of editor> /etc/containers/policy.json
```

The file that is displayed is similar to this:

```
{
  "default": [{"type": "reject"}],
  "transports": {
    "docker": {
      "quay.io": [{"type": "insecureAcceptAnything"}],
      "docker.io": [{"type": "insecureAcceptAnything"}],
      "<server-address>": [
        {
          "type": "signedBy",
          "keyType": "GPGKeys",
          "keyPath": "/tmp/containersig.txt"
        }
      ]
    }
  }
}
```

This file shows that neither `quay.io`, or `docker.io` will perform the verification, because the type is `insecureAcceptAnything` which overrides the default type of `reject`. However, `<server-address>` will perform the verification, because the parameter `type` is set to `"signedBy"`.

NOTE:

The only `keyType` currently supported is GPG keys.

2. Under the `<server-address>` entry, modify the `keyPath` `<1>` to include the name of your key file.

```

{
  "default": [{"type": "reject"}],
  "transports": {
    "docker": {
      "quay.io": [{"type": "insecureAcceptAnything"}],
      "docker.io": [{"type": "insecureAcceptAnything"}],
      "<server-address>": [{
        "type": "signedBy",
        "keyType": "GPGKeys",
        "keyPath": "/tmp/<key file name>",
        "signedIdentity": {
          "type": "matchExact"
        }
      }]
    }
  }
}

```

3. Save and close the file.

Result

- Pull the file using Podman, or your client of choice:

```
podman pull <server-address>/<container-name>:<tag name> --tls-verify=false
```

This response verifies the execution environment has been signed with no errors. If the execution environment is not signed, the command fails.

Related information

[Documentation for containers-policy.json](#)

Manage your organization's automation content

Organize, store, distribute, secure, and govern your organization's automation content, including roles, collections, and execution environments.

Ansible Automation Platform's automation hub serves as the centralized source of truth for your organization's automation assets.

Instead of developers keeping custom modules, roles, and playbooks scattered across disparate repositories or local machines, private automation hub provides a unified registry for managing Ansible Content Collections and execution environments. This eliminates duplicated effort, allowing your developers to seamlessly discover, share, and reuse established automation code.

By establishing a single repository, you significantly streamline the development lifecycle, ensuring your team is always building upon the most current and reliable foundations.

From a management and governance perspective, automation hub empowers you to enforce strict quality control and security standards before content is deployed enterprise-wide. It features robust role-based access control (RBAC) and a built-in content curation process, allowing you to establish workflows where senior developers can review, test, and approve collections before they are published to the broader organization.

Furthermore, it allows you to safely synchronize with Red Hat's public automation hub, blending certified, vendor-supported content with your team's proprietary IP in one secure location. Ultimately, automation hub helps you transform your team's output from fragmented scripts into a trusted, scalable catalog of enterprise-grade automation.

Sync automation content from remote repositories

Storing automation content in a remote repository provides centralized control, increases security, and improves consistency for automation across your organization.

Ansible Certified Content Collections are included in your subscription to Red Hat Ansible Automation Platform. Using Ansible automation hub, you can access and curate a unique set of collections from all forms of Ansible content. Red Hat Ansible content contains two types of content:

- Ansible Certified Content Collections
- Ansible validated content collections

With Ansible Certified Content Collections and Ansible validated content collections, you can build your automation library. You can update these collections manually by downloading their packages. You can use Ansible automation hub to distribute the relevant Red Hat Ansible Certified Content Collections to your users by creating a requirements file.

Use remote configurations to configure your private automation hub to synchronize with Ansible Certified Content Collections hosted on `console.redhat.com` or with your collections in Ansible Galaxy. Each remote configuration in Automation Content Remotes provides information for both the **community** and **rh-certified** repository about when the repository was **last updated**. You can add new content to Ansible automation hub at any time by using the **Edit** and **Sync** features included on the Automation Content Repositories page.

Understanding the difference between Ansible Galaxy and Ansible automation hub collections

Collections published to Ansible Galaxy are the latest content published by the Ansible community and have no joint support claims associated with them. Ansible Galaxy is the recommended front end directory for the Ansible community to access content. Collections published to Ansible automation hub are targeted to joint customers of Red Hat and selected partners. Customers need an Ansible subscription to access and download collections on Ansible automation hub. A certified collection means that Red Hat and partners have a strategic relationship in place and are ready to support joint customers. Collections might also have had additional testing and validation done against them.

Requesting a namespace on Ansible Galaxy To request a namespace through an Ansible Galaxy GitHub issue, send an email request to ansiblepartners@redhat.com that includes the GitHub username used to sign up on Ansible Galaxy. You must have logged in at least once for the system to validate. After users are added as administrators of the namespace, you can use the self-serve process to add more administrators.

Ansible Galaxy namespace naming guidelines Collection namespaces must follow Python module name convention. This means collections should have short, all lowercase names. You can use underscores in the collection name if it improves readability.

Related information

[Ansible Certified Content Collections and Ansible validated content](#)

Red Hat Certified, validated, and Ansible Galaxy content in automation hub

With Ansible automation hub, you can access and curate a unique set of collections from all forms of Ansible content.

Ansible Automation Platform offers two types of content that you can use to build your content library:

- Ansible Certified Content Collections, which are included with your Ansible Automation Platform subscription.
- Ansible validated content collections.

You can update these collections manually by downloading their packages, or sync content through console.redhat.com.

After you have configured a remote repository, use automation hub to distribute the relevant Red Hat Ansible Certified Content Collections to your users by creating a requirements file.

Related information

[Ansible Certified Content Collections and Ansible validated content](#)

[Obtain an automation hub API token](#)

[Use the API token to configure a remote repository in your local hub](#)

[Create a requirements file](#)

Configure automation hub tokens

You must create an API token before you can upload or download collections..

The automation hub API token authenticates your `ansible-galaxy` client to the Red Hat automation hub server

NOTE:

Automation hub does not support basic authentication or authenticating through service accounts. You must authenticate using token management.

Your method for creating the API token differs according to the type of automation hub that you are using:

Automation hub uses offline token management.

Private automation hub uses API token management.

Related information

[Create the offline token](#)

Create the offline token in automation hub

In automation hub, you can create an offline token using **Token management**. The offline token is a secret token used to protect your content, so be sure to store it in a secure location.

NOTE:

Your offline token expires after 30 days of inactivity.

Procedure

1. Navigate to [Ansible Automation Platform on the Red Hat Hybrid Cloud Console](#).
2. From the navigation panel, select **Automation Hub > Connect to Hub**.
3. Under **Offline token**, click **Load Token**.
4. Click the **Copy to clipboard** icon to copy the offline token.
5. Paste the token into a file and store in a secure location.

Next steps

The offline token is now available for configuring automation hub as your default collections server or for uploading collections by using the `ansible-galaxy` command line tool.

Related information

[Keeping your offline token active](#)

Refresh the offline token

Offline tokens expire after 30 days of inactivity. You can keep your offline token from expiring by refreshing it periodically.

Refreshing the offline token to keep it active is useful when an application performs an action on behalf of the user. For example, this allows the application to perform a routine data backup when the user is offline.

NOTE:

If your offline token expires, you must obtain a new one.

- Run the following command to refresh the offline token:

```
curl https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token -d grant_type=refresh_token -d client_id="cloud-services" -d refresh_token="{{ user_token }}" --fail --silent --show-error --output /dev/null
```

Related information

[Create an offline token](#)

Configure the rh-certified remote repository

You can edit the **rh-certified** remote repository to synchronize collections from automation hub hosted on console.redhat.com to your private automation hub.

Before you begin

- You have valid **Modify Ansible repo content** permissions.
- You have retrieved the Sync URL and API Token from the automation hub hosted service on console.redhat.com.
- You have configured access to port 443. This is required for synchronizing certified collections. For more information, see the automation hub table in the "Network ports and protocols" chapter of Planning your installation.

By default, your private automation hub `rh-certified` repository includes the URL for the entire group of Ansible Certified Content Collections.

To use only those collections specified by your organization, a private automation hub administrator can upload manually-created requirements files from the `rh-certified` remote.

If you have collections `A`, `B`, and `C` in your requirements file, and a new collection `X` is added to `console.redhat.com` that you want to use, you must add `X` to your requirements file for private automation hub to synchronize it.

Procedure

1. Log in to your Ansible Automation Platform.
2. From the navigation panel, select **Automation Content > Remotes**.
3. In the **rh-certified** remote repository, click **Edit remote**.
4. In the **URL** field, paste the **Sync URL**.
5. In the **Token** field, paste the token you acquired from `console.redhat.com`.
6. Click **Save remote**.

Result

You can now synchronize collections from `console.redhat.com` to your private automation hub.

Next steps

See [Synchronizing Ansible content collections in automation hub](#) for syncing steps.

Related information

[Network ports and protocols](#)

Configure the private automation hub community remote

Configure the community remote so you can sync content from Ansible Galaxy.

Before you begin

- You have a `requirements.yml` file that identifies those collections to synchronize from Ansible Galaxy as in the following example:

Requirements.yml example

```
collections:
  # Install a collection from Ansible Galaxy.
  - name: community.aws
    version: 5.2.0
    source: https://galaxy.ansible.com
```

You can edit the **community** remote repository to synchronize chosen collections from Ansible Galaxy to your private automation hub. By default, your private automation hub community repository directs to `galaxy.ansible.com/api/`.

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Automation Content > Remotes**.
3. In the **Details** tab in the **Community** remote, click **Edit remote**.
4. In the **YAML requirements** field, paste the contents of your `requirements.yml` file.
5. Click **Save remote**.

Result

You can now synchronize collections identified in your `requirements.yml` file from Ansible Galaxy to your private automation hub.

Next steps

See [Synchronizing content collections](#) for syncing steps.

Configure proxy settings

If your private automation hub is behind a network proxy, you can configure proxy settings on the remote to sync content located outside of your local network.

Before you begin

- You have valid **Modify Ansible repo content** permissions.
- You have a proxy URL and credentials from your local network administrator.

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Automation Content > Remotes**.
3. In either the **rh-certified** or **Community** remote, click the **More Actions** icon  and select **Edit remote**.
4. Expand the **Show advanced options** drop-down menu.
5. Enter your proxy URL, proxy username, and proxy password in the appropriate fields.
6. Click **Save remote**.

Related information

[Configure access to a custom automation hub repository](#)

Create a requirements file

Use a requirements file to add collections to your automation hub.

Requirements files are in YAML format and list the collections that you want to install in your automation hub. A standard `requirements.yml` file contains the following parameters:

- `name` : the name of the collection formatted as `<namespace>.<collection_name>`
- `version` : the collection version number
- Create your requirements file.

In YAML format, collection information in your requirements file should contain the following information:

```
collections:  
  - name: namespace.collection_name  
    version: 1.0.0
```

Be sure to specify the collection version number, otherwise you will sync all collection versions. Syncing all versions can require more space than expected.

Next steps

To sync the collections in your requirements file, follow the steps in [Synchronizing content collections](#).

Configure content signing on private automation hub

To successfully sign and publish Ansible Certified Content Collections, you must configure private automation hub for signing.

Before you begin

- Your GnuPG key pairs have been securely set up and managed by your organization.
- Your public-private key pair has proper access for configuring content signing on private automation hub.

Procedure

1. Create a signing script that accepts only a filename.

NOTE:

This script acts as the signing service and must generate an ascii-armored detached `gpg` signature for that file using the key specified through the `PULP_SIGNING_KEY_FINGERPRINT` environment variable.

The script prints out a JSON structure with the following format.

```
{"file": "filename", "signature": "filename.asc"}
```

All the file names are relative paths inside the current working directory. The file name must remain the same for the detached signature.

Example: The following script produces signatures for content:

```
#!/usr/bin/env bash

FILE_PATH=$1
SIGNATURE_PATH="$1.asc"

ADMIN_ID="$PULP_SIGNING_KEY_FINGERPRINT"
PASSWORD="password"

# Create a detached signature
gpg --quiet --batch --pinentry-mode loopback --yes --passphrase \
    $PASSWORD --homedir ~/.gnupg/ --detach-sign --default-key $ADMIN_ID \
    --armor --output $SIGNATURE_PATH $FILE_PATH

# Check the exit status
STATUS=$?
if [ $STATUS -eq 0 ]; then
    echo {"file\: \"$FILE_PATH\", \"signature\: \"$SIGNATURE_PATH\"}
else
    exit $STATUS
fi
```

After you deploy a private automation hub with signing enabled to your Ansible Automation Platform cluster, new UI additions are displayed in collections.

2. Review the Ansible Automation Platform installer inventory file for options that begin with `automationhub_*`.

```
[all:vars]
.
.
.

automationhub_create_default_collection_signing_service = True

automationhub_auto_sign_collections = True

automationhub_require_content_approval = True

automationhub_collection_signing_service_key = /abs/path/to/
galaxy_signing_service.gpg

automationhub_collection_signing_service_script = /abs/path/to/
collection_signing.sh
```

The two new keys (**`automationhub_auto_sign_collections`** and **`automationhub_require_content_approval`**) indicate that the collections must be signed and approved after they are uploaded to private automation hub.

Download signature public keys

After you sign and approve collections, download the signature public keys from the Ansible Automation Platform UI.

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Automation Content › Signature Keys**. The Signature Keys dashboard displays a list of multiple keys: collections and container images.
 - To verify collections, download the key prefixed with `collections-`.
 - To verify container images, download the key prefixed with `container-`.
3. Choose one of the following methods to download your public key:
 - Click the **Download Key** icon to download the public key.
 - Click the **Copy to clipboard** next to the public key you want to copy.

Result

Use the public key that you copied to verify the content collection that you are installing.

Configure Ansible-Galaxy CLI to verify collections

You can configure Ansible-Galaxy CLI to verify collections. This ensures that downloaded collections are approved by your organization and have not been changed after they were uploaded to automation hub.

Before you begin

- Public key for verification has been added to the local system keyring.
- Signed collections are available in automation hub to verify signature.
- Certified collections can be signed by approved roles within your organization.

If a collection has been signed by automation hub, the server provides ASCII armored, GPG-detached signatures to verify the authenticity of `MANIFEST.json` before using it to verify the collection's contents. First you must opt into signature verification by configuring a keyring for `ansible-galaxy` or providing the path with the `--keyring` option.

Procedure

1. To import a public key into a non-default keyring for use with `ansible-galaxy`, run the following command.

```
gpg --import --no-default-keyring --keyring ~/.ansible/pubring.kbx my-public-key.asc
```

NOTE:

In addition to any signatures provided by automation hub, signature sources can also be provided in the requirements file and on the command line. Signature sources should be URIs.

2. To verify the collection name provided on the CLI with an additional signature, run the following command:

```
ansible-galaxy collection install namespace.collection  
  
--signature https://examplehost.com/detached_signature.asc  
  
--signature file:///path/to/local/detached_signature.asc --keyring ~/.ansible/pubring.kbx
```

You can use this option multiple times to provide multiple signatures.

3. Confirm that the collections in a requirements file list any additional signature sources following the collection's signatures key, as in the following example.

```
# requirements.yml
collections:
  - name: ns.coll
    version: 1.0.0
    signatures:
      - https://examplehost.com/detached_signature.asc
      - file:///path/to/local/detached_signature.asc

ansible-galaxy collection verify -r requirements.yml --keyring ~/.ansible/
pubring.kbx
```

When you install a collection from automation hub, the signatures provided by the server are saved along with the installed collections to verify the collection's authenticity.

4. (Optional) If you need to verify the internal consistency of your collection again without querying the Ansible Galaxy server, run the same command you used previously using the `--offline` option.

Related information

[Configuring a keyring](#)

Manage collection access and permissions with namespaces

Namespaces are unique locations in automation hub to which you can upload and publish content collections.

Namespace access is governed by teams with permission to manage the content and related information that appears there. You can use namespaces in automation hub to organize collections developed within your organization for internal distribution and use.

If you are working with namespaces, you must have a team that has permissions to create, edit and upload collections to namespaces. Collections uploaded to a namespace require administrative approval before you can publish them and make them available for use.

Create a content development team

Create a new team in Ansible Automation Platform to support content curation and development in your organization.

Before you begin

- You have administrative permissions in Ansible Automation Platform and can create teams.

Your team can contribute internally-developed collections for publication in private automation hub.

To help content developers create a namespace and upload their internally developed collections to private automation hub, first create a team and assign the required permissions.

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Access Management > Teams** and click **Create team**.
3. Enter **Content Engineering** as a **Name** for the team.
4. Select an **Organization** for the team.
5. Click **Create team**. The team Details page opens.
6. Select the **Roles** tab and then select the **Automation Content** tab.
7. Click **Add roles**.
8. Select **Namespace** from the **Resource type** list and click **Next**.
9. Select the namespaces that will receive the new roles and click **Next**.
10. Select the roles to apply to the selected namespaces and click **Next**.
11. Review your selections and click **Finish**.
12. Click **Close** to complete the process.

Next steps

The new team is created with the permissions that you assigned. You can now add users to the team in the **Users** tab.

Related information

[Teams](#)

[Assign users to a team](#)

[Creating a user](#). Assigning a user to a team adds them as a member only. Use the **Roles** tab to assign a role that gives users on the team resource access.

Create a namespace

Create a namespace to organize collections that your content developers upload to automation hub.

Before you begin

- You have **Add Namespaces** and **Upload to Namespaces** permissions.

When creating a namespace, you can assign a team in automation hub as owners of that namespace.

NOTE:

By default, namespaces used on Ansible Galaxy are also used on automation hub by the Ansible partner team. For questions, contact ansiblepartners@redhat.com.

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Automation Content > Namespaces**.
3. Click **Create namespace** and enter a **Name** for your namespace.
4. Optional: enter a description, company, logo URL, resources, or useful links in the appropriate fields.
5. Click **Create namespace**.
6. Select the **Team Access** tab and click **Add roles** to assign roles to your namespace.
7. Select the team to which you want to grant a role, then click **Next**.
8. Select the roles you want to apply to the selected team, and then click **Next**.
9. Review your selections and click **Finish**.
10. Click **Close** to complete the process.

Next steps

Your content developers can now upload collections to your new namespace. In addition, teams with permissions to upload to the namespace can start adding their collections for approval. Collections in the namespace appear in the **Published** repository after approval.

Edit namespace resources

Edit the information associated with the namespace and provide resources for your users to accompany collections included in the namespace.

Before you begin

- You have **Change Namespaces** permissions.

Customize your namespace by adding a logo and a description, and link users to your GitHub repository, issue tracker, or other online assets. You can also enter markdown text in the **Resources** field to include more information. This is helpful to users who use your content in their automation tasks.

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Automation Content › Namespaces**.
3. Select the namespace you want to edit.
4. Click the **Edit namespace**.
5. Enter the relevant information in the fields.
6. Optional: enter markdown information in the **Resources** field.
7. Click **Save namespace**.

Upload collections to a namespace

Upload internally-developed collections in `tar.gz` file format to your private automation hub namespace for review and approval by an automation hub administrator.

Before you begin

- You have a namespace to which you can upload the collection.

IMPORTANT:

Attempting to upload very large collections will result in an error.

Limit collection size to 20 mb when uploading to Ansible Galaxy or console.redhat.com. For private automation hub avoid uploading collections sized 200 mb or more.

In scenarios that require a complete environment with multiple collections and dependencies, use an execution environment.

When approved, the collection moves to the **Published** content repository where automation hub users can view and download it.

NOTE:

Format your collection file name as follows: `<my_namespace-my_collection-1.0.0.tar.gz>`

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Automation Content › Namespaces** and select a namespace.
3. Select the **Collections** tab.
4. Click **Upload collection**.
5. Click **Browse** next to the **Collection file** field.
6. Select the collection to upload.

7. Select one of the following options:

- **Staging repos**
- **Repositories without pipeline**

8. Click **Upload collection**.

Result

To verify whether the collection uploaded successfully or if it failed, navigate to **Automation Content > Namespaces** and click the **More Actions** icon , then select **Imports**. There you will find a summary of tests indicating whether the import was successful.

Related information

[Pull execution environments for use in automation hub](#)

Review your namespace import logs

Review the status of collections uploaded to your namespaces to evaluate success or failure of the process.

Before you begin

- You have access to a namespace to which you can upload collections.

Imported collection information includes:

Status

completed or failed

Approval status

waiting for approval or approved

Version

the version of the uploaded collection

Import log

activities executed during the collection import

Procedure

1. Log in to your Ansible Automation Platform.
2. From the navigation panel, select **Automation Content > Namespaces**.
3. Select a namespace.
4. Click the **More Actions** icon  and select **Imports**.
5. Use the search field or locate an imported collection from the list.
6. Click the imported collection.

7. Review collection import details to determine the status of the collection in your namespace.

Delete a collection

You can further manage your collections by deleting content, if the content has no dependencies.

Before you begin

- The content being deleted does not have dependencies with other content.
- You have **Delete Collections** permissions.

The **Dependencies** tab on a collection displays a list of other collections that use the current collection.

Procedure

1. Log in to your Ansible Automation Platform.
2. From the navigation panel, select **Automation Content > Collections**.
3. Before deleting the collection, check to see if it has collections that are dependent on it:
 - Click the **Dependencies** tab for that collection. If it is blank, you will be able to delete the collection. If the **Dependencies** tab is not blank, you must delete these dependencies before you can delete the collection.
4. Click the collection to delete.
5. Click the **More Actions** icon , and then select an option:
 - a. **Delete version from system** removes the specific version of the collection from the entire instance, including all repositories and namespaces.
 - b. **Delete version from repository** removes the specific version of the collection from the repository where it was uploaded. This does not affect the collection in other repositories or namespaces.
 - c. **Delete entire collection from repository** removes all versions of the entire collection from the repository where it was uploaded, but does not affect other repositories or namespaces.
 - d. **Delete entire collection from system** removes all versions of the entire collection from the instance, including all repositories and namespaces.
6. When the confirmation window opens, verify that the collection or version number is correct, and then select **Delete**.

Delete a namespace

You can delete unwanted namespaces to manage storage on your automation hub server.

Before you begin

- The namespace you are deleting does not have a collection with dependencies.
- You have **Delete namespace** permissions.

First, ensure that the namespace you want to delete does not contain a collection with dependencies.

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Automation Content > Namespaces**.
3. Click the namespace to be deleted.
4. Click the **More Actions** icon , then click **Delete namespace**.

NOTE:

If the **Delete namespace** button is disabled, the namespace contains a collection with dependencies. Review the collections in this namespace, and delete any dependencies.

Result

The namespace that you deleted, as well as its associated collections, is now deleted and removed from the namespace list view.

Upload, group, and approve your organization's content collections

Use automation hub to manage and publish content collections developed within your organization.

Upload and group collections in namespaces. Uploaded collections need administrative approval to appear in the **Published** content repository. When the collection is published, your users can access and download it for use.

You can also reject submitted collections that do not meet organizational certification criteria.

Approve collections

You can approve collections uploaded to individual namespaces for internal publication and use.

Before you begin

- You have **Modify Ansible repo content** permissions.

All collections awaiting review are located in **Automation Content > Collection Approvals**.

Procedure

1. From the navigation panel, select **Automation Content > Collection Approvals**.
Collections requiring approval have the status **Needs review**.
2. Find the collection you want to review in the list. You can also filter collections by Namespace, Repository, and Status using the search bar.
3. Click the thumbs up icon to approve and sign the collection. Confirm your choice in the modal that appears.

Result

Approved collections are moved to the **Published** repository where users can view and download them for use.

Reject collections uploaded for review

You can reject collections uploaded to individual namespaces.

Before you begin

- You have **Modify Ansible repo content** permissions.

All collections awaiting review are located in **Automation Content > Collection Approvals**.

Collections requiring approval have the status **Needs review**.

Procedure

1. From the navigation panel, select **Automation Content > Collection Approvals**.
2. Find the collection you want to review in the list. You can also filter collections by Namespace, Repository, and Status using the search bar.
3. Click the thumbs down icon to reject the collection. Confirm your choice in the modal that appears.

Result

Collections you decline for publication are moved to the **Rejected** repository.

Repository management with automation hub

As a platform administrator, you can create, edit, delete, and move automation content collections between repositories.

Types of repositories in automation hub

In automation hub you can publish collections to two types of repositories, depending on whether you want your collection to be verified:

Staging repositories

Any user with permission to upload to a namespace can publish collections into these repositories. Collections in these repositories are not available in the search page. Instead, they are displayed on the approval dashboard for an administrator to verify. Staging repositories are marked with the `pipeline=staging` label.

Custom repositories

Any user with write permissions on the repository can publish collections to these repositories. Custom repositories can be public where all users can see them, or private where only users with view permissions can see them. These repositories are not displayed on the approval dashboard. If the repository owner enables search, the collection can appear in search results.

By default, automation hub includes one staging repository that is automatically used when a repository is not specified for uploading collections.

Approve content for custom repositories in automation hub

In automation hub you can approve collections into any repository marked with the `pipeline=approved` label.

By default, automation hub includes one repository for approved content, but you have the option to add more from the repository creation screen. You cannot directly publish into a repository marked with the `pipeline=approved` label. A collection must first go through a staging repository and be approved before being published into a `pipeline=approved` repository.

Auto approval

When auto approve is enabled, any collection you upload to a staging repository is automatically promoted to all of the repositories marked as `pipeline=approved`.

Approval required

When auto approve is disabled, the administrator can view the approval dashboard and see collections that have been uploaded into any of the staging repositories. Sorting by **Approved** displays a list of approved repositories. From this list, the administrator can select one or more repositories to which the content should be promoted.

If only one approved repository exists, the collection is automatically promoted into it and the administrator is not prompted to select a repository.

Rejection

Rejected collections are automatically placed into the rejected repository, which is pre-installed.

Manage access to custom content with role based access control

Use Role Based Access Control (RBAC) to restrict user access to custom repositories by defining access permissions based on user roles.

By default, users can view all public repositories in their automation hub, but they cannot modify a repository unless their role allows them access to do so. The same logic applies to other operations on the repository. For example, you can remove a user's ability to download content from a custom repository by changing their role permissions.

Create a custom repository

When you create a repository in Ansible Automation Platform, you can configure the repository to be private or hide it from search results.

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Automation Content > Repositories**.
3. Click **Create repository**.
4. Enter a **Name** for your repository.
5. In the **Description** field, describe the purpose of the repository.

6. To retain previous versions of your repository each time you make a change, enter a figure in the field labeled **Retained number of versions**. The number of retained versions can range anywhere between 0 and unlimited. To save all versions, leave this set to null.

NOTE:

If you have a problem with a change to your custom repository, you can [revert to a different repository version](#) that you have retained.

7. In the **Pipeline** field, select a pipeline for the repository. This option defines who can publish a collection into the repository.

Staging

Anyone is allowed to publish automation content into the repository.

Approved

Collections added to this repository are required to go through the approval process by way of the staging repository. When auto approve is enabled, any collection uploaded to a staging repository is automatically promoted to all of the approved repositories.

None

Any user with permissions on the repository can publish to the repository directly, and the repository is not part of the approval pipeline.

8. Optional: To hide the repository from search results, select **Hide from search**.
9. Optional: To make the repository private, select **Make private**. This hides the repository from anyone who does not have permissions to view the repository.
10. To sync the content from a remote repository into this repository, in the **Remote** field select the remote that contains the collections you want included in your custom repository.
11. Click **Create repository**.

Next steps

- After the repository is created, the details page is displayed.
From here, you can provide access to your repository, review or add collections, and work with the saved versions of your custom repository.

Configure access to a custom automation hub repository

Use role-based access control to assign users to a repository so they can interact with it.

By default, private repositories and the automation content collections are hidden from all users in the system. Public repositories can be viewed by all users, but cannot be modified. You must configure a custom repository to grant users access to it.

Procedure

1. Log in to private automation hub.
2. From the navigation panel, select **Automation Content > Repositories**.
3. Click into your repository in the list and select the **Team Access** tab.
4. Click **Assign teams**.
5. Select the team to which you want to grant a role, then click **Next**.
6. Select the roles you want to apply to the selected team, and then click **Next**.
7. Review your selections and click **Finish**.
8. Click **Close** to complete the process.

Related information

[Assign roles to a team](#)

Add automation content to an automation hub repository

After you create your repository, you can add automation content to it.

Procedure

1. From the navigation panel, select **Automation Content > Repositories**.
2. Click into your repository in the list.
3. Select the **Collection versions** tab.
4. Click **Add Collections** and select the collections that you want to add to your repository.
5. Click **Select**.

Manage automation hub repository versions

When automation content is added or removed from a repository, a new version is created. If a change to your repository causes a problem, you can revert to a previous version.

Reverting is a safe operation and does not delete content from the system, but rather changes the content associated with the repository. The number of versions saved is defined in the **Retained number of versions** setting when a repository is created.

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Automation Content > Repositories**.
3. Click into your repository in the list and then select the **Versions** tab.
4. Locate the version you want to revert to and click the **More Actions** icon , and select **Revert to this version**.
5. Check the box confirming your selection, and then click **Revert to repository version**.

Create a remote configuration in automation hub

Remote configurations allow you to sync content to your custom repositories from an external collection source.

In automation hub, you can create a remote configuration to an external collection source. Then, you can sync the content from those collections to your custom repositories.

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Automation Content > Remotes**.
3. Click **Create Remote**.
4. Enter a **Name** for the remote configuration.
5. Enter the **URL** for the remote server, including the path for the specific repository.

NOTE:

To find the remote server URL and repository path, navigate to **Automation Content > Repositories**, select the **More Actions** icon , and select **Copy CLI configuration**.

6. To sync signed collections only, check the box labeled **Signed collections only**.
7. To sync dependencies, check the box labeled **Sync all dependencies**. To turn off dependency syncing, leave this box unchecked.
8. Configure the credentials to the remote server by entering a **Token** or **Username** and **Password** required to access the external collection.

NOTE:

To generate a token from the navigation panel, select **Automation Content > API token**, click **Load token** and copy the token that is loaded.

9. To access collections from console.redhat.com, enter the **SSO URL** to sign in to the identity provider (IdP).
10. Select or create a **Requirements file** to identify the collections and version ranges to synchronize with your custom repository. For example, to download only the kubernetes and AWS collection versions 5.0.0 or later the requirements file would look like this:

```
Collections:
- name: community.kubernetes
- name: community.aws
  version: ">=5.0.0"
```

11. Optional: To configure your remote further, use the options available under **Show advanced options**:
 - a. If there is a corporate proxy in place for your organization, enter a **Proxy URL**, **Proxy Username** and **Proxy Password**.
 - b. Enable or disable transport layer security using the **TLS validation** checkbox.
 - c. If digital certificates are required for authentication, enter a **Client key** and **Client certificate**.
 - d. If you are using a self-signed SSL certificate for your server, enter the PEM encoded client certificate used for authentication in the **CA certificate** field.
 - e. To accelerate the speed at which collections in this remote can be downloaded, specify the number of collections that can be downloaded in tandem in the **Download concurrency** field.
 - f. To limit the number of queries per second on this remote, specify a **Rate Limit**.

NOTE:

Some servers can have a specific rate limit set, and if exceeded, synchronization fails.

Assign access to a remote configuration

After you create a remote configuration, you must provide access to it before anyone can use it.

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Automation Content > Remotes**.
3. Click into your repository in the list, and then select the **Team Access** tab.
4. Click **Add roles**.

5. Select the team to which you want to grant a role, then click **Next**.
6. Select the roles you want to apply to the selected team, and then click **Next**.
7. Review your selections and click **Finish**.
8. Click **Close** to complete the process.

Synchronize repositories in automation hub

Synchronize repositories in automation hub to update your custom repository with the latest collections from a configured remote.

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Automation Content > Repositories**.
3. Locate your repository in the list and click **More Actions** icon , then select **Sync repository**.

All collections in the configured remote are downloaded to your custom repository. To check the status of the collection sync, select **Automation Content > Task Management** from the navigation panel.

NOTE:

To limit repository synchronization to specific collections within a remote, you can identify specific collections to be pulled by using a `requirements.yml` file.

Related information

[Create a requirements file](#)

Export automation content from automation hub

When collections are finalized, you can import them to a location where they can be distributed to others across your organization.

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Automation Content > Collections**. The **Collections** page displays all collections across all repositories. You can search for a specific collection.
3. Click into the collection that you want to export. The collection details page opens.

4. From the **Install** tab, select **Download tarball**. The .tar file is downloaded to your default browser downloads folder. You can now import it to the location of your choosing.

Import automation content to automation hub

As an automation content creator, you can import a collection to your custom repository.

To use a collection in your custom repository, first import the collection into your namespace so the automation hub administrator can approve it.

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Automation Content > Namespaces**. The **Namespaces** page displays all of the namespaces available.
3. Select the namespace to which you want to add your collection.
4. Select the **Collections** tab.
5. Click **Upload Collection**.
6. Enter or browse to select a collection file.
7. Select the repository pipeline to add the collection. The choices are **Staging repos** and **Repositories without pipeline**.
8. Click **Upload collection**.

The **Imports** screen displays a summary of tests and notifies you if the collection upload is successful or has failed. To find your imports, on your namespace click the **More Actions** icon  and select **Imports**.

NOTE:

If the collection is not approved, it is not displayed in the published repository.

Related information

[Approval pipeline for custom repositories in automation hub](#)

Manage containers in your private automation hub

Private automation hub functions as an internal container registry for your organization. It allows you to store, manage, and govern the container images, or automation execution environments, that your teams use to run automation.

Private automation hub functions as an internal container registry for your organization. It allows you to store, manage, and govern the container images, or automation execution environments, that your teams use to run automation.

To effectively manage these containers, first learn the difference between the two types of registries in your workflow:

- External registries (the source): Public or third-party registries where you source your initial images. Common examples include the Red Hat Ecosystem Catalog (registry.redhat.io) or Quay.io. You can pull images from these registries to your local environment.
- Private automation hub registry, or your remote registry (the destination): Your internal, secure registry hosted on private automation hub. You push your curated and approved images here. Your Ansible Automation Platform infrastructure then pulls these images from private automation hub to execute jobs.

Populate your private automation hub container registry

The automation hub remote registry is used for storing and managing automation execution environments.

When you have built or sourced an execution environment, you can push that execution environment to the registry portion of private automation hub to create a container repository. Then, you can grant your team access to the container repository, and customize the repository with a README, relevant links and other information for your team's use.

The workflow is as follows:

1. Pull an execution environment from an external registry (like registry.redhat.io) to your local machine.
2. Tag the image locally for your private automation hub registry.
3. Push the image to your private automation hub.
4. Configure access permissions and documentation (such as READMEs) within private automation hub so your teams can use the image.

IMPORTANT:

As of **April 1st, 2025**, `quay.io` is adding three additional endpoints. As a result, you must adjust the allow/block lists within your firewall systems lists to include the following endpoints:

- `cdn04.quay.io`
- `cdn05.quay.io`
- `cdn06.quay.io`

To avoid problems pulling container images, customers must allow outbound TCP connections (ports 80 and 443) to the following hostnames:

- `cdn.quay.io`
- `cdn01.quay.io`
- `cdn02.quay.io`
- `cdn03.quay.io`
- `cdn04.quay.io`
- `cdn05.quay.io`
- `cdn06.quay.io`

This change should be made to any firewall configuration that specifically enables outbound connections to `registry.redhat.io` or `registry.access.redhat.com`.

Use the hostnames instead of IP addresses when configuring firewall rules.

After making this change, you can continue to pull images from `registry.redhat.io` or `registry.access.redhat.com`. You do not require a `quay.io` login, or need to interact with the `quay.io` registry directly in any way to continue pulling Red Hat container images.

Related information

[Firewall changes for container image pulls 2024/2025](#)

[Network ports and protocols](#)

Pull execution environments for use in automation hub

Before you can push execution environments to your private automation hub, you must first pull them from an existing registry and tag them for use.

Before you begin

- You have permissions to pull automation execution environments from `registry.redhat.io`.

The following example details how to pull an execution environment from the Red Hat Ecosystem Catalog (`registry.redhat.io`).

Procedure

1. Log in to Podman with your `registry.redhat.io` credentials:

```
$ podman login registry.redhat.io
```

2. Pull an execution environment:

```
$ podman pull registry.redhat.io/<ee_name>:<tag>
```

Result

To verify that the execution environment you pulled is contained in the list, take these steps:

1. List the images in local storage:

```
$ podman images
```

2. Check the execution environment name, and verify that the tag is correct.

Related information

[Red Hat Ecosystem Catalog Help](#)

Tag container images

Tag automation execution environments to add an additional name to automation execution environments stored in your automation hub container repository. If no tag is added to an automation execution environment, automation hub defaults to `latest` for the name.

Before you begin

- You have **change automation execution environment tags** permissions.

Procedure

1. From the navigation panel, select **Automation Content > Execution Environments**.
2. Select your automation execution environments.
3. Click the **Images** tab.

4. Click the **More Actions** icon , and click **Manage tags**.
5. Add a new tag in the text field and click **Add**.
6. Optional: Remove **current tags** by clicking **x** on any of the tags for that image.

Result

- Click the **Activity** tab and review the latest changes.

Pull and sync images from automation hub to your local system

Pull Ansible Automation Platform execution environments from the automation hub registry to your local machine. Use the provided `podman pull` command for the `latest` version in the repository, or specify a tag to copy a specific execution environment.

Pull an image

Use the user interface to pull an execution environment from your private automation hub remote registry to make a copy to your local machine.

Before you begin

- You must have permission to view and pull from a private container repository.
- If you are pulling automation execution environments from a password or token-protected registry, [create a credential](#) first.

Procedure

1. From the navigation panel, select **Automation Content > Execution Environments**.
2. Select your execution environment.
3. In the **Pull this image** entry, click **Copy to clipboard**.
4. Paste and run the command in your terminal.

Result

- Run `podman images` to view images on your local machine.

Sync images from a container registry

You can pull automation execution environments from the private automation hub remote registry to sync an image to your local machine. To sync an execution environment from a remote registry, you must first configure a remote registry.

Before you begin

You must have permission to view and pull from a private container repository.

Procedure

1. From the navigation panel, select **Automation Content > Execution Environments**.
2. Add <https://registry.redhat.io> to the registry.
3. Add any required credentials to authenticate.

NOTE:

Some remote registries are aggressive with rate limiting. Set a rate limit under **Advanced Options**.

4. From the navigation panel, select **Automation Content > Execution Environments**.
5. Click **Create execution environment** in the page header.
6. Select the registry you want to pull from. The **Name** field displays the name of the automation execution environments displayed on your local registry.

NOTE:

The **Upstream name** field is the name of the image on the remote server. For example, if the upstream name is set to "alpine" and the **Name** field is "local/alpine", the alpine image is downloaded from the remote and renamed to "local/alpine".

7. Set a list of tags to include or exclude. Note that syncing automation execution environments with a large number of tags is time consuming and uses a lot of disk space.

Related information

[Red Hat Container Registry Authentication](#)

[What is Podman?](#)

Create a credential

To pull automation execution environments images from a password or token-protected registry, you must create a credential.

In earlier versions of Ansible Automation Platform, you were required to deploy a registry to store execution environment images. On Ansible Automation Platform 2.0 and later, the system operates as if you already have a remote registry up and running. To store execution environment images, add the credentials of only your selected remote registries.

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Automation Execution > Infrastructure > Credentials**.
3. Click **Create credential** to create a new credential.
4. Enter an authorization **Name**, **Description**, and **Organization**.
5. In the **Credential Type** drop-down, select **Container Registry**.
6. Enter the **Authentication URL**. This is the remote registry address.
7. Enter the **Username** and **Password or Token** required to log in to the remote registry.
8. Optional: To enable SSL verification, select **Verify SSL**.
9. Click **Create credential**.

Remote registry team permissions

Configure team access to container repositories in private automation hub to control who can access and manage execution environments.

New teams do not have any assigned permissions by default. You must add permissions when first creating a team or edit an existing team to add or remove permissions.

The following table lists permissions you can grant to teams to ensure they have the correct level of access and privileges to your remote registries.

Team permissions for managing containers in private automation hub

Permission name	Description
Create new containers	Users can create new containers
Change container namespace permissions	Users can change permissions on the container repository
Change container	Users can change information on a container

Permission name	Description
Change execution environment tags	Users can modify execution environment tags
Push to existing container	Users can push an execution environment to an existing container

Related information
[Assign roles to a team](#)

Add a README to your container repository

Add a README to your container repository to provide instructions to your users on how to work with the container. Automation hub container repositories support Markdown for creating a README. By default, the README is empty.

Before you begin

- You have permissions to change containers.

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Automation Content > Execution Environments**.
3. Select your execution environment.
4. On the **Detail** tab, click **Add**.
5. In the **Raw Markdown** text field, enter your README text in Markdown.
6. Click **Save** when you are finished.

Next steps

After you add a README, you can edit it at any time by clicking **Edit** and repeating steps 4 and 5.

Trigger automation from events with Event-Driven Ansible

Event-Driven Ansible is a highly scalable, flexible automation capability. It works with external event sources (like software vendors' monitoring tools) to identify IT events and automatically implement the defined changes or responses within a rulebook.

Learn how to configure and manage the end-to-end lifecycle of event-driven automation within the Red Hat Ansible Automation Platform. Event-Driven Ansible provides you the ability to perform the following actions:

- Manage projects for rulebook storage
- Build and restore decision environments for execution
- Define credentials for secure authentication
- Create event streams for centralized routing
- Manage rulebook activations to main persistent, scalable listeners that monitor infrastructure and trigger automated workflows in real time

NOTE:

- API documentation for Event-Driven Ansible controller is available at <https://<gateway-host>/api/eda/v1/docs>
- To meet high availability demands, Event-Driven Ansible controller shares centralized Redis (REmote DIctionary Server) with the Ansible Automation Platform UI. When Redis is unavailable, you will not be able to create or sync projects, or enable rulebook activations.

IMPORTANT:

In new installations of Ansible Automation Platform, using Event-Driven Ansible controller's API to manage organizations, teams, or users requires an automated sync of up to 15 minutes to propagate changes to the rest of the Ansible Automation Platform components. To avoid potential errors and ensure immediate access, use the platform gateway API instead, or the unified UI.

Related information

[Configure central authentication for Ansible Automation Platform](#)

[Manage access with role-based access control](#)

[Trigger automation from events with Event-Driven Ansible](#)

Organize rulebooks for event-driven automation

Projects are a logical collection of rulebooks. They must be a git repository and located in the path defined for Event-Driven Ansible content in Ansible collections: `/extensions/eda/rulebooks` at the root of the project.

IMPORTANT:

To meet high availability demands, Event-Driven Ansible controller shares centralized Redis (REmote DIctionary Server) with the Ansible Automation Platform UI. When Redis is unavailable, you will not be able to create or sync projects.

Related information

[Remote Dictionary Server \(Redis\)](#)

Set up a new project

Set up a project to connect Event-Driven Ansible controller to a Git repository, enabling it to pull, sync, and manage the rulebooks used by your automation.

Before you begin

- You are logged in to the Ansible Automation Platform Dashboard as a Content Consumer.
- You have set up a credential, if necessary. For more information, see the [Setting up credentials](#) section.
- You have an existing repository containing rulebooks.

Procedure

1. Log in to the Ansible Automation Platform Dashboard.
2. Navigate to **Automation Decisions > Projects**.
3. Click **Create project**.
4. Insert the following:

Name

Enter project name.

Description

This field is optional.

Source control type

Git is the only source control type available for use. This field is optional.

Source control URL

Enter Git, SSH, or HTTP[S] protocol address of a repository, such as GitHub or GitLab. This required field is editable. See [Editing a project](#) to view details of how editing this field impacts rulebook activations.

NOTE:

This field accepts SSH private key or private key phrase. To enable the use of these private keys, your project URL must begin with `git@`.

Proxy

This is used to access HTTP or HTTPS servers. This field is optional and editable. See [Editing a project](#) to view details of how editing this field impacts rulebook activations.

Source control branch/tag/commit

This is the branch to checkout. In addition to branches, you can input tags, commit hashes, and arbitrary refs. Some commit hashes and refs may not be available unless you also provide a custom refspec. This field is optional and editable. See [Editing a project](#) to view details of how editing this field impacts rulebook activations.

Source control refspec

A refspec to fetch (passed to the Ansible git module). This parameter allows access to references via the branch field not otherwise available. This field is optional and editable. See [Editing a project](#) to view details of how editing this field impacts rulebook activations.

Source control credential

This is an optional credential used to authenticate with the provided Source control URL.

Content signature validation credential

Enable content signing to verify that the content has remained secure when a project is synced. If the content has been tampered with, the job will not run. This field is optional.

Options

The Verify SSL option is enabled by default. Enabling this option verifies the SSL with HTTPS when the project is imported.

NOTE:

You can disable this option if you have a local repository that uses self-signed certificates.

5. Select **Create project**.

Result

Your project is now created and can be managed in the **Projects** page.

After saving the new project, the project's details page is displayed. From there or the **Projects** list view, you can edit or delete it.

Projects list view

The Projects list view provides a quick dashboard to verify rulebook sync status, track content via the Git hash, and manage repository access.

NOTE:

If a rulebook changes in source control, you can re-sync a project by selecting the sync icon next to the project from the **Projects** list view. The **Git hash** updates represent the latest commit on that repository. An activation must be restarted or recreated if you want to use the updated project.

Edit a project

Edit project details (like the source repository URL or credential) to manage where Event-Driven Ansible fetches its rulebooks from and how it authenticates.

Procedure

1. From the **Projects** list view, select the **More Actions** icon  next to the desired project. The Edit page is displayed.
2. Edit the desired fields.

IMPORTANT:

When you update a project's **Source control URL**, **Source control branch/tag/commit**, or **Source control refspect**, Event-Driven Ansible automatically triggers a project resync. This process updates the rulebooks available within Event-Driven Ansible controller and can significantly impact existing rulebook activations:

- **Rulebook Content Updates:** Running activations continue to use old content when a rulebook's content changes. To apply the newer content, you must restart the affected rulebook activation. If the rulebook content you update is attached to an activation that uses event streams, you must re-attach the event stream to that activation after the updates are applied and then, restart the activation.
- **New Rulebooks:** Any new rulebook added to the repository becomes available in the database after the sync.
- **Deleted Rulebooks:** A removed rulebook is deleted from the database upon sync. Its associated activations, however, continue to run and can be restarted. Review and update any activations detached from their source rulebook.

3. Select **Save project**.

Delete a project

Use the platform gateway interface to delete a project that is no longer required from Event-Driven Ansible controller.

Before you begin

1. A project is not currently linked to any rulebook activation.

NOTE:

You cannot delete a project if it is still being used by a rulebook activation.

Procedure

1. To delete a project, complete one of the following:
 - From the **Projects** list view, select the checkbox next to the desired project, and click the **More Actions** icon  from the page menu.
 - From the **Projects** list view, click the **More Actions** icon  next to the desired project.
2. Select **Delete project**.
3. In the **Permanently delete projects** window, select **Yes, I confirm that I want to delete this project**.
4. Select **Delete project**.

Create runtime environments for event-driven automation

Decision environments are container images that run Ansible rulebooks. They create a common language for communicating automation dependencies, and give a standard way to build and distribute the automation environment.

You can find the default decision environment in the *Ansible Rulebook*.

To create your own decision environment, see *Install Ansible Builder to create or edit execution environments* and *Build a custom decision environment for Event-Driven Ansible*. Refer to the Related Links section for more details.

Related information

[Ansible Rulebook](#)[Install Ansible Builder to create or edit execution environments](#)[Build a custom decision environment for Event-Driven Ansible](#)

Supported event sources and filters

The capabilities of your decision environment are largely defined by the event sources and filters included in your container image. Event sources serve as the foundation for {EDAcontroller}, determining where your rulebooks receive incoming signals.

Selecting compatible sources is critical for successful deployment, as availability can vary—some sources are exclusive to the web-based Event-Driven Ansible controller, while others are optimized for the `ansible-rulebook` command-line interface (CLI).

When building your decision environments, the `de-minimal` image provides a curated set of supported plugins designed to ensure stability and compatibility. However, as the ecosystem evolves, plugin namespaces have been updated to reflect their specific collection origins (for example, migrating from `ansible.eda` to specific cloud collections). To ensure your rulebooks remain functional, verify that your definitions align with the latest plugin naming conventions.

The following table lists the supported plugins in the `de-minimal` image, including their plugin type and any associated deprecated names:

Supported event sources and filters

Plugin name	Event filter	Deprecated name
<code>eda.builtin.dashes_to_underscores</code>	Event filter	<code>ansible.eda.dashes_to_underscores</code>
<code>eda.builtin.event_splitter</code>	Event filter	
<code>eda.builtin.generic</code>	Event source	<code>ansible.eda.generic</code>
<code>eda.builtin.insert_hosts_to_meta</code>	Event filter	<code>ansible.eda.insert_hosts_to_meta</code>
<code>eda.builtin.insert_meta_info</code>	Event filter	
<code>eda.builtin.json_filter</code>	Event filter	<code>ansible.eda.json_filter</code>
<code>eda.builtin.normalize_keys</code>	Event filter	<code>ansible.eda.normalize_keys</code>
<code>eda.builtin.pg_listener</code>	Event source	<code>ansible.eda.pg_listener</code>
<code>eda.builtin.range</code>	Event source	<code>ansible.eda.range</code>

Plugin name	Event filter	Deprecated name
<code>eda.builtin.webhook</code>	Event source	<code>ansible.eda.webhook</code>

The following event sources have been made available in dedicated collections. The original `ansible.eda` names are currently supported as deprecated plugins in `de-minimal`, but you must transition to the new `de-supported` versions to ensure long-term compatibility and overall improvements:

de-supported event sources

Event source (in <code>de-supported</code>)	Deprecated name (in <code>de-minimal</code>)
<code>amazon.aws.aws_cloudtrail</code>	<code>ansible.eda.aws_cloudtrail</code>
<code>amazon.aws.aws_sqs_queue</code>	<code>ansible.eda.aws_sqs_queue</code>
<code>azure.azcollection.azure_service_bus</code>	<code>ansible.eda.azure_service_bus</code>

In addition, the `de-minimal` image currently provides event sources that are not supported, and will be removed in a future release:

- `ansible.eda.file`
- `ansible.eda.file_watch`
- `ansible.eda.journald`
- `ansible.eda.tick`
- `ansible.eda.url_check`

Build a custom decision environment for Event-Driven Ansible

Customize a decision environment container image to ensure your rulebook activations run with the precise, custom-maintained collections and dependencies they require.

Before you begin

- Ansible Automation Platform > = 2.5
- Event-Driven Ansible
- Ansible Builder > = 3.0

IMPORTANT:

- Use the correct Event-Driven Ansible controller decision environment in Ansible Automation Platform to prevent rulebook activation failure.
 - If you want to connect Event-Driven Ansible controller to Ansible Automation Platform 2.4, you must use `registry.redhat.io/ansible-automation-platform-24/de-minimal-rhel9:latest` (recommended) or `registry.redhat.io/ansible-automation-platform-24/de-minimal-rhel8:latest`
 - If you want to connect Event-Driven Ansible controller to Ansible Automation Platform 2.5, you must use `registry.redhat.io/ansible-automation-platform-25/de-minimal-rhel9:latest` (recommended) or `registry.redhat.io/ansible-automation-platform-25/de-minimal-rhel8:latest`
 - If you want to connect Event-Driven Ansible controller to Ansible Automation Platform 2.6, you must use `registry.redhat.io/ansible-automation-platform-26/de-minimal-rhel9:latest`

Procedure

1. Use `de-minimal` as the base image with Ansible Builder to build your custom decision environments. This image is built from a base image provided by Red Hat at [Ansible Automation Platform minimal decision environment](#).

IMPORTANT: The `ansible.eda` collection is already installed in the `de-minimal` base image. To prevent Ansible Builder from attempting to reinstall it, add `ansible.eda` to the `exclude.all_from_collections` list as shown in the following examples.

The following is an example of the Ansible Builder definition file that uses `de-minimal` as a base image to build a custom decision environment with the `ansible.eda` collection:

```
version: 3

images:
  base_image:
    name: 'registry.redhat.io/ansible-automation-platform-25/de-minimal-
rhel9:latest'

dependencies:
  galaxy:
    collections:
      - name: servicenow.itsm
  python_interpreter:
    package_system: "python3.12"
  exclude:
    all_from_collections:
      # ansible.eda is already installed in de-minimal
      - ansible.eda

options:
  package_manager_path: /usr/bin/microdnf
```

2. Optional: If you need other Python packages or RPMs, add the following to a single definition file:

```

version: 3

images:
  base_image:
    name: 'registry.redhat.io/ansible-automation-platform-25/de-minimal-
rhel9:latest'

dependencies:
  galaxy:
    collections:
      - name: servicenow.itsm
  python:
    - six
    - psutil
  python_interpreter:
    package_system: "python3.12"
  exclude:
    all_from_collections:
      # ansible.eda is already installed in de-minimal
      - ansible.eda

options:
  package_manager_path: /usr/bin/microdnf

```

Set up a new decision environment

Set up a new decision environment to define the dedicated, containerized runtime (including collections and dependencies) necessary to execute your rulebook activations.

Before you begin

- You have set up a credential, if necessary. For more information, see the [Setting up credentials](#) section.
- You have pushed a decision environment image to an image repository or you chose to use the `de-minimal` image located in [registry.redhat.io](#).

Procedure

1. Log in to Ansible Automation Platform.
2. Navigate to **Automation Decisions > Decision Environments**.
3. Click **Create decision environment**.
4. Insert the following:

Name

Insert the name.

Description

This field is optional.

Organization

Select an organization to associate with the decision environment.

Image

This is the full image location, including the container registry, image name, and version tag.

Pull

This optional field defines when the Event-Driven Ansible controller attempts to download (pull) the specified container image from the registry before running an automation rulebook. This setting is crucial for ensuring security, consistency, and efficient use of system resources by controlling when the local image cache is updated or reused.

- **Always pull container before running** - checks the registry for a new version of the image before every use, even if a local copy exists. This ensures that the freshest content is always used, which is critical for security patches, content updates, and staging environments.
- **Only pull the image if not present before running** - checks the local cache of Event-Driven Ansible controller first. If the image is found locally, it is used immediately. A pull from the registry only happens if the image is not found locally. This optimizes performance by avoiding unnecessary downloads and is best for production environments where the content image is stable and faster startup is required.
- **Never pull container before running** - never attempts to pull the image from the registry. It only uses an image that has been previously pulled and is present in the local cache. This option guarantees consistency by preventing unintended content changes. It requires the image to be manually pulled to Event-Driven Ansible controller beforehand. If the image isn't present, the activation fails.

Credential

This field is optional. This is the credential needed to use the decision environment image.

5. Select **Create decision environment**.

Result

Your decision environment is now created and can be managed on the **Decision Environments** page.

After saving the new decision environment, the decision environment's details page is displayed. From there or the **Decision Environments** list view, you can edit or delete it.

Define rules that trigger automation from events

A rulebook is a set of conditional rules that Event-Driven Ansible uses to perform IT actions in an event-driven automation model. Rulebooks are the means that users tell Event-Driven Ansible which source to check for an event and when that event occurs what to do when certain conditions are met.

A rulebook specifies actions to be performed when a rule is triggered. A rule gets triggered when the events match the conditions for the rules. The following actions are currently supported:

- `run_playbook` (only supported with ansible-rulebook CLI)
- `run_module`
- `run_job_template`
- `run_workflow_template`
- `set_fact`
- `post_event`
- `retract_fact`
- `print_event`
- `shutdown`
- `debug`
- `none`

To view further details, see *Ansible Actions*.

A rulebook activation is a process running in the background defined by a decision environment executing a specific rulebook. You can set up your rulebook activation by following *Set up a rulebook activation*.

WARNING:

Red Hat does not recommend the use of a non-supported source plugin with 1 postgres database. This can pose a potential risk to your use of Ansible Automation Platform.

IMPORTANT:

To meet high availability demands, Event-Driven Ansible controller shares centralized *Redis* (*REmote DIctionary Server*) with the Ansible Automation Platform UI. When Redis is unavailable, the following functions will not be available:

- Creating an activation, if `is_enabled` is True
- Deleting an activation
- Enabling an activation, if not already enabled
- Disabling an activation, if not already disabled
- Restarting an activation

Related information

[Set up a rulebook activation](#)

[Ansible Actions](#)

[Remote Dictionary Server \(Redis\)](#)

Set up a rulebook activation

Create a rulebook activation to link a rulebook to a decision environment and event sources, initiating the event-driven automation process.

Before you begin

- You are logged in to the Ansible Automation Platform Dashboard as a Content Consumer.
- You have set up a project.
- You have set up a decision environment.

Procedure

1. Log in to Ansible Automation Platform.
2. Navigate to the **Automation Decisions > Rulebook Activations**.
3. Click **Create rulebook activation**.
4. Insert the following:

Name

Insert the name.

Description

This field is optional.

Organization

Enter your organization name or select Default from the list.

Project

Projects are a logical collection of rulebooks. This field is optional.

Rulebook

Rulebooks are displayed according to the project selected.

Credential

Select 0 or more credentials for this rulebook activation. This field is optional.

NOTE:

- The credentials that display in this field are customized based on your rulebook activation and only include the following credential types: Vault, Red Hat Ansible Automation Platform, or any custom credential types that you have created. For more information about credentials, see [Credentials](#).
- If you plan to use a Red Hat Ansible Automation Platform credential, you can *only* select 1 Red Hat Ansible Automation Platform credential type for a rulebook activation.

Decision environment

Decision environments are a container image to run Ansible rulebooks.

NOTE:

In Event-Driven Ansible controller, you cannot customize the pull policy of the decision environment. By default, it follows the behavior of the **always** policy. Every time an activation is started, the system tries to pull the most recent version of the image.

Restart policy

This is the policy that determines how an activation should restart after the container process running the source plugin ends.

- Policies:
 - a. **Always:** This restarts the rulebook activation immediately, regardless of whether it ends successfully or not, and occurs no more than 5 times.
 - b. **Never:** This never restarts a rulebook activation when the container process ends.
 - c. **On failure:** This restarts the rulebook activation after 60 seconds by default, only when the container process fails, and occurs no more than 5 times.

Log level

This field defines the severity and type of content in your logged events.

- Levels:
 - a. **Error:** Logs that contain error messages that are displayed in the **History** tab of an activation.
 - b. **Info:** Logs that contain useful information about rulebook activations, such as a success or failure, triggered action names and their related action events, and errors.
 - c. **Debug:** Logs that contain information that is only useful during the debug phase and might be of little value during production. This log level includes both error and log level data.

Service name

This defines a service name for Kubernetes to configure inbound connections if the activation exposes a port. This field is optional.

Rulebook activation enabled?

This automatically enables the rulebook activation to run.

Variables

The variables for the rulebook are in a JSON or YAML format. The content would be equivalent to the file passed through the `--vars` flag of `ansible-rulebook` command.

NOTE:

In the context of automation controller and Event-Driven Ansible controller, you can use both `extra_vars` and credentials to store a variety of information. However, credentials are the preferred method of storing sensitive information such as passwords or API keys because they offer better security and centralized management, whereas `extra_vars` are more suitable for passing dynamic, non-sensitive data.

Options

Check the **Skip audit events** option if you do not want to see your events in the Rule Audit.

5. Click **Create rulebook activation**.

Result

Your rulebook activation is now created and can be managed on the **Rulebook Activations** page.

After saving the new rulebook activation, the rulebook activation's details page is displayed, with either a **Pending**, **Running**, or **Failed** status. From there or the **Rulebook Activations** list view, you can restart or delete it.

NOTE:

Occasionally, when a source plugin shuts down, it causes a rulebook to exit gracefully after a certain amount of time. When a rulebook activation shuts down, any tasks that are waiting to be performed will be canceled, and an info level message is sent to the activation log. For more information, see [Rulebooks](#).

Rulebook activation list view

Use the **Rulebook Activations** list view to quickly monitor the operational status, event fire count, and restart frequency of all your deployed automation services.

If the **Status** is **Running**, it means that the rulebook activation is running in the background and executing the required actions according to the rules declared in the rulebook.

You can view more details by selecting the activation from the **Rulebook Activations** list view.

For all activations that have run, you can view the **Details** and **History** tabs to get more information about what happened.

View activation output

View the activation output in the History tab to inspect the execution results, confirm successful automation runs, and diagnose runtime errors or task failures.

Procedure

1. Select the **History** tab to access the list of all the activation instances. An activation instance represents a single execution of the activation.
2. Then select the activation instance you want to view. The **Output** for the activation instance is displayed.

Next steps

To view events that came in and triggered an action, navigate to **Automation Decisions > Rule Audit** and follow instructions in the [Rule Audit](#) section.

Manage active event-driven automation integrations

To maintain efficient event-driven workflows, you must manage the lifecycle of your rulebook activations. Managing these activations effectively ensures that your automation remains responsive, reduces resource duplication, and minimizes downtime during configuration updates.

Enable and disable rulebook activations

Toggle the state of an activation to start or stop event processing instantly, allowing for temporary pauses, maintenance, or troubleshooting without deletion.

Procedure

1. Select the switch on the row level to enable or disable your chosen rulebook.
2. In the window, select **Yes, I confirm that I want to enable/disable these X rulebook activations.**
3. Select **Enable/Disable rulebook activation.**

Restart rulebook activations

Restart an activation to immediately apply configuration changes, update rulebook content, or quickly recover from unexpected failures.

NOTE:

You can only restart a rulebook activation if it is currently enabled and the restart policy was set to **Always** when it was created.

Procedure

1. Select the **More Actions** icon  next to **Rulebook Activation enabled/disabled** toggle.
2. Select **Restart rulebook activation.**
3. In the window, select **Yes, I confirm that I want to restart these X rulebook activations.**
4. Select **Restart rulebook activations.**

Edit a rulebook activation

Update a rulebook activation's settings after deployment to adjust parameters, update variables, or troubleshoot an active automation failure

Procedure

1. On the Rulebook Activations page, next to the activation you want to edit, toggle the **Rulebook Activation enabled** button to the off position first to disable the activation.
The **Disable rulebook activations** message is displayed asking you to confirm that you want to disable the activation.
2. Select the **Yes, I confirm that I want to disable these <1> rulebook activations** checkbox and click **Disable rulebook activations**.
3. Next to the rulebook activation, click the **Edit** icon. This takes you to the Edit form.

NOTE:

You can also access the **Edit** feature by clicking the rulebook activation on the Rulebook Activations page, toggling the **Rulebook activation enabled** button to the off position, confirming that you want to disable the activation, and clicking the **Edit rulebook activation** button on the top right of the page to access the Edit form.

4. Edit the desired fields.

NOTE:

If you prefer to run your activation immediately, you can toggle the **Rulebook activation enabled** button to the on position, and then save your changes.

5. Click **Save rulebook activation**.

Result

This takes you back to the Rulebook Activations page.

Delete rulebook activations

End and permanently remove a rulebook activation and its configuration when its automated event-driven workflow is no longer required.

Procedure

1. Select the **More Actions** icon  next to the **Rulebook Activation enabled/disabled** toggle.
2. Select **Delete rulebook activation**.
3. In the window, select **Yes, I confirm that I want to delete these X rulebook activations**.
4. Select **Delete rulebook activations**.

Reuse event-driven automation configurations

Streamline your workflow by duplicating existing rulebook activations to reuse proven configurations. This section covers how to create copies, manage unique naming requirements, and navigate bulk-enablement safety warnings to avoid duplicate job failures.

Duplicate a rulebook activation

When setting up a new rulebook activation with field inputs that are similar to one of your existing rulebook activations, you can use the **Duplicate rulebook activation** feature instead of manually entering input into each field.

While setting up rulebook activations can be a lengthy process, the ability to duplicate the required fields from an existing activation saves time and, in some cases, reduces the possibility of human error.

Procedure

1. On the Rulebook Activations page, click the **More Actions** icon  on the row of the activation you want to duplicate. The More Actions list is displayed with three options:

- **Restart rulebook activation**
- **Duplicate rulebook activation**
- **Delete rulebook activation**

2. Select **Duplicate rulebook activation**.

A message is displayed: "<Name of rulebook activation 1> duplicated." Initially, the newly duplicated activation is displayed as disabled on the Rulebook Activations page with the same name as the original activation followed by a time stamp in 24-hour format (for example, <Name of rulebook activation 1> @ 18:43:27).

IMPORTANT:

The original rulebook activation continues to run after you have duplicated it. If you try to enable the duplicated activation without editing the fields (including the Name field) to distinguish it from the original, a message is displayed reminding you that the rulebook activation was duplicated from an original, and enabling it might fail or result in duplicate jobs and other complications.

3. Before you run the duplicated rulebook activation, edit the fields by completing the following:
 - a. Next to the duplicated rulebook activation, click the **Edit** icon. This takes you to the Edit form.
 - b. Edit the desired fields.

NOTE:

Ensure that you have given your newly duplicated activation a meaningful **Name** that distinguishes it from the original activation.

4. Toggle the **Enable rulebook activation** button to the on position.
5. After confirming all of your edits are complete, click **Save rulebook activation**.

Result

This initiates the rulebook activation, and if it runs successfully, the status changes to **Running** or **Completed**.

Activate webhook-based automation in Openshift

In Openshift environments, you can activate webhooks by creating a route to expose the activation's service, enabling external systems to send events and trigger automation.

Before you begin

- You have created a rulebook activation.

NOTE:

The following is an example of rulebook with a given webhook:

```
- name: Listen for storage-monitor events
hosts: all
sources:
  - ansible.eda.webhook:
      host: 0.0.0.0
      port: 5000
rules:
  - name: Rule - Print event information
    condition: event.meta.headers is defined
    action:
      run_job_template:
        name: StorageRemediation
        organization: Default
        job_args:
          extra_vars:
            message: from eda
            sleep: 1
```

Procedure

1. Create a Route (on OpenShift Container Platform) to expose the service. The following is an example Route for an ansible-rulebook source that expects POST's on port 5000 on the decision environment pod:

```

kind: Route
apiVersion: route.openshift.io/v1
metadata:
  name: test-sync-bug
  namespace: dynatrace
  labels:
    app: eda
    job-name: activation-job-1-5000
spec:
  host: test-sync-bug-dynatrace.apps.aap-dt.ocp4.testing.ansible.com
  to:
    kind: Service
    name: activation-job-1-5000
    weight: 100
  port:
    targetPort: 5000
  tls:
    termination: edge
    insecureEdgeTerminationPolicy: Redirect
  wildcardPolicy: None

```

2. When you create the Route, test it with a **Post to the Route URL**:

NOTE:

You do not need the port as it is specified on the Route (targetPort).

```

curl -H "Content-Type: application/json" -X POST
test-sync-bug-dynatrace.apps.aap-dt.ocp4.testing.ansible.com -d
'{}'

```

Testing with OpenShift (Kubernetes)

Configure Kubernetes networking (for example, Ingress) to temporarily expose activation webhooks for non-production testing and debugging.

Procedure

1. Run the following command to expose the port on the cluster for a given service:

```
kubectll port-forward svc/<ACTIVATION_SVC_NAME> 5000:5000
```

2. Make the HTTP requests against the `localhost:5000` to trigger the rulebook:

```
curl -H "Content-Type: application/json" -X POST test-sync-bug-dynatrace.apps.aap-dt.ocp4.testing.ansible.com -d '{}'
```

Audit pipeline-triggered automation activity

Rule audit allows the auditing of rules which have been triggered by all the rules that were activated at some point.

The **Rule Audit** list view shows you a list of every time an event came in that matched a condition within a rulebook and triggered an action. The list shows you rules within your rulebook and each heading matches up to a rule that has been executed.

View rule audit details

View the rule audit details to quickly identify the rulebook activation responsible, the event that matched the condition, and the overall execution history of the rule.

Procedure

1. From the navigation panel select **Automation Decisions > Rule Audit**.
2. Select the desired rule, this brings you to the **Details** tab.

Result

From here you can view when it was created, when it was last fired, and the rulebook activation that it corresponds to.

View rule audit events

Review a rule's event history to inspect the payload, source type, and timestamp of the raw event data that matched the condition and triggered the rule.

Procedure

1. From the navigation panel, select **Automation Decisions > Rule Audit**.
2. Select the desired rule, this brings you to the **Details** tab. To view all the events that triggered an action, select the **Events** tab. This shows you the event that triggered actions.
3. Select an event to view the **Event log**, along with the **Source type** and **Timestamp**.

View rule audit actions

Review the output of executed actions within a triggered rule to confirm that the desired automated response was initiated and completed successfully.

Procedure

1. From the navigation panel select **Automation Decisions > Rule Audit**.
2. Select the desired rule, then select the **Actions** tab.

Result

From here, you can view executed actions that were taken. Some actions are linked out to Automation Execution where you can view the output.

Troubleshoot failed event-driven automation triggers

Rulebook activations might occasionally fail due to various reasons. While many issues can be resolved through basic checks, diagnosing failures across a distributed system requires robust logging.

Event-Driven Ansible's enhanced logging strategy includes the addition of unique tracking identifiers to all output to significantly improve troubleshooting.

Review the list of possible issues contained in this chapter that can cause activation failures and suggestions on how you can resolve them. For detailed log filtering using the new identifiers, see Event-Driven Ansible log filtering.

Related information

[Event-Driven Ansible log filtering](#)

Event-Driven Ansible log filtering

Event-Driven Ansible includes tracking identifiers in all log output to significantly improve troubleshooting. These identifiers help track user actions and activation processes across multiple services and log files.

Event-Driven Ansible log tracking IDs

Identifier	Abbreviation	Purpose	Location
X-REQUEST-ID	<code>rid</code>	Tracks HTTP requests from the platform gateway through the entire Event-Driven Ansible request lifecycle. Use this to correlate UI actions or API calls with backend processing.	Included in the HTTP response headers and Event-Driven Ansible log entries.
Log Tracking ID	<code>tid</code>	Tracks the activation lifecycle from creation through completion, persisting across restarts and multiple log files.	Included in all activation-related log entries. It can be obtained from the activation History tab in the UI.
Activation Instance ID	<code>aiid</code>	Identifies the logs specific to a single execution instance of a rulebook activation, allowing you to view <code>ansible-rulebook</code> output for that run.	Included in activation logs.

NOTE:

Not all processes originate from a user or external client. When an Event-Driven Ansible orchestrator internally triggers a process (for example, a monitor request), the `rid` UUID is generated internally to track that process lifecycle and will not be present in the platform gateway logs.

The enhanced log format places these identifiers at the start of the message, making them easy to filter:

```
`[rid: <UUID>] [tid: <UUID>] [aaid: <ID>] aap_eda.tasks.orchestrator Processing request...`
```

Use log filtering for troubleshooting

Learn to filter logs using specialized tracking identifiers for efficient troubleshooting of activation issues and API request lifecycles.

Procedure

1. Collect identifiers:

- a. When an issue occurs, retrieve the **Log Tracking ID** (`tid`) from the failed activation instance's logs in the UI **History tab**.
- b. If the issue was triggered by a user action (like restarting an activation), obtain the **X-REQUEST-ID** (`rid`) from the HTTP response headers.

2. Search system logs:

- a. Use the collected UUID to search through your backend logs (worker, scheduler, API, and the like.). This filters out irrelevant noise, allowing you to focus on the full timeline of the specific request or activation across all services.

3. Correlate timeline:

- a. Use the common `tid` to follow the activation's progress (or failure) across different log files and services.

4. Use support tools:

- a. If necessary, use `sosreport` or `mustgather` tools, which automatically collect all relevant Event-Driven Ansible logs from `/var/log/ansible-automation-platform/eda/`.

Resolve rulebook activations stuck in pending state

Diagnose and resolve issues preventing a rulebook activation from transitioning from Pending to a running, operational state.

Procedure

1. Confirm whether there are other running activations and if you have reached the limits (for example, memory or CPU limits).
 - a. If there are other activations running, terminate one or more of them, if possible.
 - b. If not, check that the default worker, Redis, and activation worker are all running.
 - c. If all systems are working as expected, check your eda-server internal logs in the worker, scheduler, API, and nginx containers and services to see if the problem can be determined. These logs reveal the source of the issue, such as an exception thrown by the code, a runtime error with network issues, or an error with the rulebook code. If your internal logs do not provide information that leads to resolution, report the issue to Red Hat support.

Fix rulebook activations stuck in a restart loop

Troubleshoot rulebook activations that restart repeatedly (indicating persistent errors) to diagnose and fix core issues preventing stable, continuous automation execution.

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Automation Decisions > Rulebook Activations**.
3. From the **Rulebook Activations** page, select the activation in your list that keeps restarting. The Details page is displayed.
4. Click the **History** tab for more information and select the rulebook activation that keeps restarting. The Details tab is displayed and shows the output information.
5. Check the **Restart policy** field for your activation.

There are three selections available: **On failure** (restarts a rulebook activation when the container process fails), **Always** (always restarts regardless of success or failure with no more than 5 restarts), or **Never** (never restarts when the container process ends).

- a. Confirm that your rulebook activation Restart policy is set to **On failure**. This is an indication that an issue is causing it to fail.

- b. To possibly diagnose the problem, check the YAML code and the instance logs of the rulebook activation for errors.
 - c. If you cannot find a solution with the restart policy values, proceed to the next steps related to the **Log level**.
6. Check your log level for your activation.
 - a. If your default log level is **Error**, go back to the **Rulebook Activation** page and recreate your activation following procedures in [Setting up rulebook a activation](#).
 - b. Change the **Log level** to **Debug**.
 - c. Run the activation again and navigate to the **History** tab from the activation details page.
 - d. On the **History** page, click one of your recent activations and view the **Output**.

Resolve event processing failures in rulebook activations

Troubleshoot why a running rulebook activation is failing to process events, focusing on common causes like source definition mismatches or internal processing errors.

Procedure

1. **Check the rulebook source:** Review the source plugin defined in your rulebook YAML (for example, `ansible.eda.webhook`, `ansible.eda.kafka`).
2. **Verify event input:** Confirm that the events you are sending to Event-Driven Ansible controller are compatible with the source plugin defined in the rulebook. If the rulebook expects a Kafka message, it cannot process a generic webhook event.
3. **Confirm activation mapping:** If you are using event streams, ensure the correct event stream is mapped to the rulebook during the activation setup. A mismatch here will result in the activation receiving no data.

Troubleshoot actions that fail to trigger after receiving events

If your rulebook activation is **Running** and successfully receiving events, but no actions are being executed, the issue is likely within the logic of your rulebook.

Procedure

1. **Check rule conditions:** Review the rulebook YAML to confirm that the conditions (the when statements) are accurately written and precisely match the structure and values of the incoming event payload.
2. **Verify indentation and syntax:** Ensure all rulebook syntax and indentation are correct, as a simple error can prevent the rule engine from evaluating conditions.
3. **Validate actions:** Confirm that the specified action is a recognized and correctly configured action (for example, `run_job_template` with the proper arguments).

Troubleshoot event streams that fail to send events

Diagnose issues where an event stream is receiving data but failing to forward it, ensuring proper connectivity and correct credential setup.

- Try the following options to resolve this.
 - a. Ensure that each of your event streams in Event-Driven Ansible controller is *not* in **Test** mode . This means activations would not receive the events.
 - b. Verify that the origin service is sending the request properly.
 - c. Check that the network connection to your platform gateway instance is stable. If you have set up event streams, this is the entry of the event stream request from the sender.
 - d. Verify that the proxy in the platform gateway is running.
 - e. Confirm that the event stream worker is up and running, and able to process the request.
 - f. Verify that your credential is correctly set up in the event stream.
 - g. Confirm that the request complies with the authentication mechanism determined by the set credential (for example, basic must contain a header with the credentials or HMAC must contain the signature of the content in a header, and similar).

NOTE:

The credentials might have been changed in Event-Driven Ansible controller, but not updated in the origin service.

- h. Verify that the rulebook that is running in the activation reacts to these events. This would indicate that you wrote down the event source *and* added actions that consume the events coming in. Otherwise, the event does reach the activation but there is nothing to activate it.
- i. If you are using self-signed certificates, you might want to disable certificate validation when sending webhooks from vendors. Most of the vendors have an option to disable certificate validation for testing or non-production environments.

Respond to events from external systems

Simplified event routing provides Event-Driven Ansible controller the capability to capture and analyze data from various remote systems (like GitHub or GitLab) using event streams. You can attach one or more event streams to an activation by swapping out sources in a rulebook.

Event streams simplify connecting sources to rulebooks. This capability enables the creation of a single endpoint to receive alerts from an event source for utilization in multiple rulebooks.

Event streams

Event streams provide the secure, authenticated entry point for external systems to send events over the internet directly to Event-Driven Ansible controller, simplifying remote data ingestion.

Event-Driven Ansible controller supports six different event stream types.

Event Stream Types

Type	Description	Vendor examples
Hashed Message Authentication Code (HMAC)	Uses a shared secret between Event-Driven Ansible controller and the vendors webhook server. This guarantees message integrity.	Github
Basic Authentication	Uses HTTP basic authentication.	Datadog, Dynatrace
Token Authentication	Validates incoming event data using a security token passed in the request header. While the standard HTTP header used is Authorization , it can be customized for specific platforms, such as using X-Gitlab-Token for GitLab integrations.	Gitlab, ServiceNow
OAuth2	Uses Machine-to-Machine (M2M) mode with a grant type called client	Dynatrace

Type	Description	Vendor examples
	credentials . The token is opaque.	
OAuth2 with JWT	Uses M2M mode with a grant type called client credentials . The token is JSON Web Token (JWT).	Datadog
Elliptic Curve Digital Signature Algorithm (ECDSA)	Verifies message authenticity using a public/private key pair. The sender signs the message with a private key, and the receiver (Event-Driven Ansible controller) validates it with a public key.	SendGrid, Twilio
Mutual Transport Layer Security (mTLS)	Ensures two-way authentication between Event-Driven Ansible controller and the client sending events through an event stream. It has two sub-types: • Certificate • Subject Needs the vendor's CA certificate to be present in our servers at startup. This supports non-repudiation.	PagerDuty

IMPORTANT:

If you are using an mTLS event stream with a load balancer, you must enable SSL pass-through (or L4 routing) in your load balancer configuration.

This is required because the SSL termination and client certificate validation for mTLS must occur at the platform gateway proxy server. Consult your load balancer documentation for details on enabling SSL pass-through.

Event-Driven Ansible controller also supports four other specialized event streams that are based on the six basic event stream types:

- GitLab event stream
- GitHub event stream

- ServiceNow event stream
- Dynatrace event stream

These specialized types limit the parameters you use by adding default values. For example, the GitHub event stream is a specialization of the HMAC event stream with many of the fields already populated. After the GitHub event stream credential has been saved, the recommended defaults for this event stream are displayed.

Create an event stream credential

Create a credential to establish the authentication mechanism (like basic auth or HMAC) required for external systems to securely send events to an event stream.

Before you begin

- Each event stream must have exactly one credential.

Procedure

1. Log in to the Ansible Automation Platform Dashboard.
2. From the navigation panel, select **Automation Decisions > Infrastructure > Credentials**.
3. Click **Create credential**.
4. Insert the following:

Name

Insert the name.

Description

This field is optional.

Organization

Click the list to select an organization or select **Default**.

Credential type

Click the list to select your Credential type.

NOTE:

When you select the credential type, the **Type Details** section is displayed with fields that are applicable for the credential type you selected.

Type Details

Add the requested information for the credential type you selected. For example, if you selected the GitHub Event Stream credential type, you are required to add an HMAC

Secret (symmetrical shared secret) between Event-Driven Ansible controller and the remote server.

5. Click **Create credential**.

Result

The Details page is displayed. From there or the **Credentials** list view, you can edit or delete it.

Create an event stream

Create a dedicated stream endpoint to simplify how external systems send events, making it easier to route data to multiple rulebook activations.

Before you begin

- If you will be attaching your event stream to a rulebook activation, ensure that your activation has a decision environment and project already set up.
- If you plan to connect to automation controller to run your rulebook activation, ensure that you have created a Red Hat Ansible Automation Platform credential type in addition to the decision environment and project. For more information, see [Setting up a Red Hat Ansible Automation Platform credential](#).

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Automation Decisions > Event Streams**.
3. Click **Create event stream**.
4. Insert the following:

Name

Insert the name.

Organization

Click the list to select an organization or select **Default**.

Event stream type

Select the event stream type you prefer.

NOTE:

This list displays at least 10 default event stream types that can be used to authenticate the connection coming from your remote server.

Credentials

Select a credential from the list, preferably the one you created for your event stream.

Headers

Enter HTTP header keys, separated by commas, that you want to include in the event payload.

IMPORTANT:

If your automation relies on HTTP headers being present in the event payload, you must explicitly define them to avoid unintentional exposure of sensitive information. For more information about HTTP headers and how to securely configure them, see [HTTP headers](#) and [Configuring HTTP headers securely for event streams](#).

Forward events to rulebook activation

Use this option to enable or disable the capability of forwarding events to rulebook activations.

NOTE:

The event stream's event forwarding can be disabled for testing purposes while diagnosing connections and evaluating the incoming data. Disabling the **Forward events to rulebook activation** option allows you to test the event stream connection with the remote system, analyze the header and payload, and if necessary, diagnose credential issues. This ensures that events are not be forwarded to rulebook activations causing rules and conditions to be triggered inadvertently while you are in test mode. Some enterprises might have policies to change secrets and passwords at regular cadence. You can enable/disable this option anytime after the event stream is created.

5. Click **Create event stream**.

Result

After creating your event stream, the following outputs occur:

- The Details page is displayed. From there or the Event Streams list view, you can edit or delete it. Also, the Event Streams page shows all of the event streams you have created and the following columns for each event: **Events received**, **Last event received**, and **Event stream type**. As the first two columns receive external data through the event stream, they are continuously updated to let you know they are receiving events from remote systems.
- If you disabled the event stream, the Details page is displayed with a warning message, **This event stream is disabled**.

NOTE:

After an event stream is created, the associated credential cannot be deleted until the event stream it is attached to is deleted.

- Your new event stream generates a URL that is necessary when you configure the webhook on the remote system that sends events.

HTTP headers

In the context of Event-Driven Ansible and event streams, HTTP headers play a significant role because they carry the necessary context and security information about the incoming event from a third-party source (for example, GitHub, a monitoring tool, or a proprietary webhook).

They include the following capabilities:

Authentication and non-repudiation

This is the most critical use. Headers often contain tokens, API keys, or security signatures (like an HMAC in an `X-Hub-Signature` header) that Event-Driven Ansible uses to *verify the sender's identity* and ensure the event payload has not been tampered with. This supports non-repudiation—proof that the event came from a legitimate source.

Debugging and Logging

Headers provide crucial data points (`Date` , `User-Agent` , `X-Request-ID`) for tracing the event's path, helping system administrators and SREs *debug* issues related to delayed or failed event processing.

Headers are essential for all HTTP communication, serving several distinct purposes:

- **Context and metadata:** Describe the data being sent (for example, `Content-Type: application/json`, `Content-Length: 1024`).
- **Client/Server Capabilities:** Inform the receiving party of the sender's capabilities or preferences (for example, `Accept-Language: en-US`).
- **Authentication/Authorization:** Carry security credentials (for example, `Authorization: Bearer <token>`).
- **Caching:** Controls how content should be cached by clients and proxies (for example, `Cache-Control: max-age=3600`).
- **Routing and Tracking:** They facilitate network routing and transaction tracking, often via custom headers (for example, `X-Request-ID`).

Configuring HTTP headers securely for event streams

To enhance event stream security, you must explicitly define which HTTP headers are passed. These headers carry the critical context and authentication data required for processing.

Procedure

1. To include all HTTP headers, enter an asterisk (*) in the Headers field. This allows all HTTP headers with the exception of a few headers:

- **Excluded:** Headers that begin with `X-Envoy` , `X-Trusted-Proxy` , `X-Forwarded-For` , and `X-Real-Id`
- **Redacted:** Authorization header (for example, `Authorization: Redacted`)

IMPORTANT:

If the **Headers** field is empty, none of the HTTP headers will be added to the event payload in Production and Test mode.

2. To include a specific set of HTTP headers, enter the names of the desired headers as a comma-delimited string (for example, `Host,Authorization,X-Request-ID`).

Static Unique Universal Identifiers (UUIDs) for event streams

You can configure an event stream with a static Unique Universal Identifier (UUID) to ensure its webhook URL remains consistent, even if the event stream service is recreated.

This feature is relevant for disaster recovery scenarios where external systems, like firewalls or third-party webhooks, cannot be easily reconfigured to use a new URL. Here are key concepts when considering using static UUIDs:

Disaster recovery support

A static UUID ensures that the external webhook URL, which follows the format, `https://your-eda-server.com/api/eda/v1/external_event_stream/{uuid}/` , does not change upon service recreation.

Uniqueness

The UUID you provide must be unique across all existing event streams in the system.

Security warning

Static UUIDs make your webhook URLs predictable and, therefore, could reduce security. Only use this feature when URL consistency is critical and you have implemented strong additional security measures (like strong authentication and network restrictions). For normal operations, always use autogenerated (dynamic) UUIDs.

You must ensure that additional security measures are in place, such as robust credential types (HMAC, mTLS) and network restrictions.

Update an event stream with a static UUID (API Method)

Access the API to set a static UUID, a feature critical for maintaining webhook URL consistency across service recreations, such as in disaster recovery scenarios.

Before you begin

- Ansible Automation Platform 2.6-next

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Overview**.
3. In the URL, replace **Overview** with the API endpoint, `api/eda/v1/` (for example, `https://<platformURL>.com/api/eda/v1/`). The `Api V1 Root` page displays a list of Event-Driven Ansible resource URLs.

```

{
  "config": "http://<aap.platform>.com/api/eda/v1/config/",
  "status": "http://<aap.platform>/api/eda/v1/status/",
  "openapi-json": "http://<aap.platform>/api/eda/v1/openapi.json",
  "openapi-yaml": "http://<aap.platform>/api/eda/v1/openapi.yaml",
  "openapi-docs": "http://<aap.platform>/api/eda/v1/docs/",
  "openapi-redoc": "http://<aap.platform>/api/eda/v1/redoc/",
  "session-login": "http://<aap.platform>/api/eda/v1/auth/session/login/",
  "session-logout": "http://<aap.platform>/api/eda/v1/auth/session/logout/",
  "token-refresh": "http://<aap.platform>/api/eda/v1/auth/token/refresh/",
  "current-user": "http://<aap.platform>/api/eda/v1/users/me/",
  "project-list": "http://<aap.platform>/api/eda/v1/projects/",
  "rulebook-list": "http://<aap.platform>/api/eda/v1/rulebooks/",
  "activation-list": "http://<aap.platform>/api/eda/v1/activations/",
  "activationinstance-list": "http://<aap.platform>/api/eda/v1/activation-
instances/",
  "auditrule-list": "http://<aap.platform>/api/eda/v1/audit-rules/",
  "user-list": "http://<aap.platform>/api/eda/v1/users/",
  "controller-token-list": "http://<aap.platform>/api/eda/v1/users/me/awx-
tokens/",
  "credentialtype-list": "http://<aap.platform>/api/eda/v1/credential-
types/",
  "edacredential-list": "http://<aap.platform>/api/eda/v1/eda-credentials/",
  "credentialinputsource-list": "http://<aap.platform>/api/eda/v1/
credential-input-sources/",
  "decisionenvironment-list": "http://<aap.platform>/api/eda/v1/decision-
environments/",
  "organization-list": "http://<aap.platform>/api/eda/v1/organizations/",
  "team-list": "http://<aap.platform>/api/eda/v1/teams/",
  "eventstream-list": "http://<aap.platform>/api/eda/v1/event-streams/"
}

```

4. At the end of the list, click the `"eventstream-list"` URL. This takes you to the **Event Stream List** page.
5. Locate and copy the `"id"` of the event stream UUID that you want to update. This can be found at the end of the event stream data.

```

},
"id": *1*,
"owner": "admin",
"url": "https://<platformURL>/eda-event-streams/api/eda/v1/
external_event_stream/bab8dddd-51cc-424f-87a4-0ed8ebe0a755/post/",
"created_at": "2025-11-18T16:30:45.622363Z",
"modified_at": "2025-11-18T16:30:45.622374Z",
"test_content_type": "",
"test_content": "",
"test_error_message": "",
"test_headers": "",
"events_received": 0,
"last_event_received_at": null

```

6. Paste the id at the end of the URL (for example, `https://<platformURL>.com/api/eda/v1/<id#>`) and press **Enter**. The **Event Stream Instance** page is displayed, including the current `"uuid"` value.
7. In the form at the end of the **Event Stream Instance** page, update the value of the `"uuid"` field to a unique static string of your choice.
8. Click **Patch**.

Result

- Confirm that the UUID of your event stream has been updated to the new static string.

Configure your remote system to send events

After you have created your event stream, you must configure your remote system to send events to Event-Driven Ansible controller. The method used for this configuration varies, depending on the vendor for the event stream credential type you select.

Before you begin

- The URL that was generated when you created your event stream
- Secrets or passwords that you set up in your event stream credential

The following example demonstrates how to configure webhooks in a remote system like GitHub to send events to Event-Driven Ansible controller. Each vendor will have unique methods for configuring your remote system to send events to Event-Driven Ansible controller.

Procedure

1. Log in to your GitHub repository.
2. Click **Your profile name** → **Your repositories**.

NOTE:

If you do not have a repository, click **New** to create a new one, select an owner, add a **Repository name**, and click **Create repository**.

3. Navigate to **Settings** (tool bar).
4. In the **General** navigation pane, select **Webhooks**.
5. Click **Add webhook**.
6. In the **Payload URL** field, paste the URL you saved when you created your event stream.
7. Select **application/json** in the **Content type** list.
8. Enter your **Secret**.
9. Click **Add webhook**.

Result

After the webhook has been added, it attempts to send a test payload to ensure there is connectivity between the two systems (GitHub and Event-Driven Ansible controller). If it can successfully send the data, you will see a green check mark next to the **Webhook URL** with the message, **Last delivery was successful**.

Verify your event streams work

Confirm end-to-end event flow by verifying the event stream receives data from the remote system, validating the webhook URL and authentication setup.

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Automation Decisions > Event Streams**.
3. Select the event stream that you created to validate connectivity and ensure that the event stream sends data to the rulebook activation.
4. Verify that the events were received. The number of **Events received** is displayed along with a header that contains details about the event.

If you scroll down in the UI, you can also see the body of the payload with more information about the webhook.

The **Header** and **Body** sections for the event stream are displayed on the Details page. They differ based on the vendor who is sending the event. The header and body can be used to check the attributes in the event payload, which will help you in writing conditions in your rulebook.

5. Toggle the **Forward events to rulebook activation** option to enable you to push your events to a rulebook activation.

Result

This moves the event stream to production mode and makes it easy to attach to rulebook activations. When this option is toggled off, your ability to forward events to a rulebook activation is disabled and the **This event stream is disabled** message is displayed.

Replace sources and attach event streams to activations

Replace complex source mappings with pre-configured event streams to simplify rulebook activation design and centralize incoming event routing.

There are several key points to keep in mind regarding source mapping:

1. An event stream can only be used once in a rulebook source swap. If you have multiple sources in the rulebook, you can only replace each source once.
2. The source mapping happens only in the current rulebook activation. You must repeat this process for any other activations using the same rulebook.
3. The source mapping is valid only if the rulebook doesn't get modified. If the rulebook gets modified during the source mapping process, the source mapping would fail and it would have to be repeated.
4. If the rulebook is modified after the source mapping has been created and a **Restart** happens, the rulebook activation fails.

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Automation Decisions > Rulebook Activations**.
3. Click **Create rulebook activation**.
4. Insert the following:

Name

Insert the name.

Description

This field is optional.

Organization

Enter your organization name or select Default from the list.

Project

Projects are a logical collection of rulebooks. This field is optional.

NOTE:

Although this field is optional, selecting a project helps refine your list of rulebooks choices.

Rulebook

Rulebooks are shown according to the project selected. Select a rulebook.

NOTE:

After you have selected a rulebook, the Event streams field is enabled. You can click the gear icon to display the Event streams mapping form.

Event streams

All the event streams available and set up to forward events to rulebook actiavtions are displayed. If you have not created any event streams, this field remains disabled.

Click the gear icon to display the Event streams mapping UI.

Event streams ✕

Event streams are server-side webhooks that enable you to connect various event sources to your rulebook activations. To add event streams to your rulebooks, replace an `ansible.eda.webhook` or compatible custom source with the desired event stream. This modifies the activation only, while leaving your filters intact.

Rulebook	Number of sources
demo_rulebook.yml	1

Mapping 1 ✕

Rulebook source * ⓘ
 Demo Source ▼

Event stream * ⓘ
 test ▼

Preview of source from rulebook

```
name: Demo Source
ansible.eda.generic:
  payload:
```

Save Cancel

Complete the following fields:

Rulebook source

A rulebook can contain multiple sources across multiple rulesets. You can map the same rulebook in multiple activations to multiple event streams. While managing event streams, unnamed sources are assigned temporary names (`__SOURCE {n}`) for identification purposes.

Select `__SOURCE_1` from the list.

Event stream

Select your event stream name from the list.

Click **Save**.

Event streams can replace matching sources in a rulebook, and are server-side webhooks that enable you to connect various event sources to your rulebook activations. Source types that can be replaced with the event stream's source of type `ansible.eda.pg_listener` include `ansible.eda.webhook` and other compatible webhook source plugins. Replacing selected sources affects this activation only, and modifies the rulebook's source type, source name, and arguments. Filters, rules, conditions, and actions are all unaffected.

You can select which source you want to replace with a single event stream. If there are multiple sources in your rulebook, you can choose to replace each one of them with event streams, but you are not required to replace each one. The following image displays which sources can be replaced.

```

1  ---
2  - name: Github Webhook Ruleset
3    hosts: all
4    sources:
5      - name: my_github
6        ansible.eda.webhook:
7          port: 5555
8          host: 0.0.0.0
9        filters:
10         - ansible.eda.json_filter:
11             include_keys:
12               - payload
13               - repository
14               - name
15               - full_name
16               - url
17               - sender
18               - login
19               - zen
20             exclude_keys:
21               - 'x'
22         rules:
23           - name: Webhook event
24             condition: true
25             action:
26               debug:
27
28
29  - name: SNOW Webhook Ruleset
30    hosts: all
31    sources:
32      - name: my_snow
33        ansible.eda.webhook:
34          port: 5556
35          host: 0.0.0.0
36        filters:
37         - ansible.eda.json_filter:
38             include_keys:
39               - payload
40               - approval
41               - sys_class_name
42               - sys_created_by
43               - category
44             exclude_keys:
45               - urgency
46             - 'x'
47         rules:
48           - name: Snow webhook event
49             condition: true
50             action:
51               debug:

```

The items in pink demonstrate the sources that can be replaced: source type, source name, and arguments. The remaining items (filters, rules, and actions) are not replaced.

Credential

Select 0 or more credentials for this rulebook activation. This field is optional.

NOTE:

The credentials that display in this field are customized based on your rulebook activation and only include the following credential types: Vault, Red Hat Ansible Automation Platform, or any custom credential types that you have created. For more information on credentials, see [Credentials](#).

Decision environment

A decision environment is a container image used to run Ansible rulebooks.

NOTE:

In Event-Driven Ansible controller, you cannot customize the pull policy of the decision environment. By default, it follows the behavior of the always policy. Every time an activation is started, the system tries to pull the most recent version of the image.

Restart policy

This is the policy that determines how an activation should restart after the container process running the source plugin ends.

- Policies:
 - a. **Always:** This restarts the rulebook activation immediately, regardless of whether it ends successfully or not, and occurs no more than 5 times.
 - b. **Never:** This never restarts a rulebook activation when the container process ends.
 - c. **On failure:** This restarts the rulebook activation after 60 seconds by default, only when the container process fails, and occurs no more than 5 times.

Log level

This field defines the severity and type of content in your logged events.

- Levels:
 - a. **Error:** Logs that contain error messages that are displayed in the **History** tab of an activation.
 - b. **Info:** Logs that contain useful information about rulebook activations, such as a success or failure, triggered action names and their related action events, and errors.
 - c. **Debug:** Logs that contain information that is only useful during the debug phase and might be of little value during production. This log level includes both error and log level data.

Service name

This defines a service name for Kubernetes to configure inbound connections if the activation exposes a port. This field is optional.

Rulebook activation enabled?

This automatically enables the rulebook activation to run.

Variables

The variables for the rulebook are in a JSON or YAML format. The content would be equivalent to the file passed through the `--vars` flag of `ansible-rulebook` command.

Options

Check the **Skip audit events** option if you do not want to see your events in the Rule Audit.

5. Click **Create rulebook activation**.

Result

After you create your rulebook activation, the **Details** page is displayed. You can navigate to the **Event streams** page to confirm your events have been received.

Filter events before triggering automation

Events sometimes have extra data that is unnecessary and might overwhelm the rule engine. Use event filters to remove that extra data so you can focus on what matters to your rules. Event filters might also change the format of the data so that the rule conditions can better match the data.

Events are defined as python code and distributed as collections. The default EDA collection has the following filters:

Name	Description
json_filter	This filter includes and excludes keys from the event object
dashes_to_underscores	This filter changes the dashes in all keys in the payload to be underscore
ansible.eda.insert_hosts_to_meta	This filter is used to add host information into the event so that ansible-rulebook can locate it and use it
ansible.eda.normalize_keys	This filter is used if you want to change non alpha numeric keys to underscore

You can chain event filters one after the other, and the updated data is sent from one filter to the next. Event filters are defined in the rulebook after a source is defined. When the rulebook starts the source plugin it associates the correct filters and transforms the data before putting it into the queue.

```
sources:
  - name: azure_service_bus
    ansible.eda.azure_service_bus:
      conn_str: "{{connection_str}}"
      queue_name: "{{queue_name}}"
    filters:
      - json_filter:
          include_keys: ['clone_url']
          exclude_keys: ['*_url', '_links', 'base', 'sender', 'owner', 'user']
      - dashes_to_underscores:
```

In this example the data is first passed through the `json_filter` and then through the `dashes_to_underscores` filter. In the event payload, keys can only contain letters, numbers, and underscores. The period (.) is used to access nested keys.

Since every event should record the origin of the event the filter `eda.builtin.insert_meta_info` is added automatically by ansible-rulebook to add the `source name`, `type`, and `received_at`. The `received_at` stores a date time in UTC ISO8601 format and includes the microseconds. The `uuid` stores the unique id for the event. The `meta` key is used to store metadata about the event and its needed to correctly report about the events in the aap-server.

Related information

[EDA Event Filters](#)

Author event filters

You can create custom Python event filters to clean, normalize, or enrich incoming event data, ensuring it meets the format and quality required for rulebook condition evaluation.

The basic structure follows:

```
# my_namespace.my_collection/extensions/eda/plugins/event_filter/my_filter.py
def main(event: dict, arg1, arg2):
    # Process event data here
    return event
```

You can use this filter in a rulebook by adding it to the filters list in an event source:

```
sources:
- name: azure_service_bus
  ansible.eda.azure_service_bus:
    conn_str: "{{connection_str}}"
    queue_name: "{{queue_name}}"
  filters:
  - my_namespace.my_collection.my_filter:
    arg1: hello
    arg2: world
```

Related information

[ansible.eda collection](#)

Use awx-manage to access automation controller information

Use the `awx-manage` utility to access detailed internal information of automation controller. Commands for `awx-manage` must run as the `awx` user only.

Inventory Import

`awx-manage` is a mechanism by which an automation controller administrator can import inventory directly into automation controller.

To use `awx-manage` properly, you must first create an inventory in automation controller to use as the destination for the import.

For help with `awx-manage`, run the following command:

```
awx-manage inventory_import [--help]
```

The `inventory_import` command synchronizes an automation controller inventory object with a text-based inventory file, dynamic inventory script, or a directory of one or more, as supported by core Ansible.

When running this command, specify either an `--inventory-id` or `--inventory-name`, and the path to the Ansible inventory source (`--source`).

```
awx-manage inventory_import --source=/ansible/inventory/ --inventory-id=1
```

By default, inventory data already stored in automation controller blends with data from the external source.

To use only the external data, specify `--overwrite`.

To specify that any existing hosts get variable data exclusively from the `--source`, specify `--overwrite_vars`.

The default behavior adds any new variables from the external source, overwriting keys that already exist, but preserving any variables that were not sourced from the external data source.

```
awx-manage inventory_import --source=/ansible/inventory/ --inventory-id=1 --  
overwrite
```

NOTE:

Edits and additions to Inventory host variables persist beyond an inventory synchronization as long as `--overwrite_vars` is not set.

Cleanup of old data

`awx-manage` has a variety of commands used to clean old data from automation controller.

Automation controller administrators can use the automation controller **Management Jobs** interface for access or use the command line.

- `awx-manage cleanup_jobs [--help]`
This permanently deletes the job details and job output for jobs older than a specified number of days.
- `awx-manage cleanup_activitystream [--help]`
This permanently deletes any activity stream data older than a specific number of days.

Cluster management

This section describes how to manage a automation controller cluster by provisioning and deprovisioning cluster instances. Automation controller uses the `awx-manage` command-line tool to manage cluster instances.

For more information about the `awx-manage provision_instance` and `awx-manage deprovision_instance` commands, see [Clustering](#).

IMPORTANT:

Do not run other `awx-manage` commands unless instructed by Ansible Support.

Analytics gathering

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

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

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

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

The `--since` parameter is optional.

The `--json` flag specifies the output format and is optional.

Red Hat product documentation legal notices

Copyright © Red Hat

Except as otherwise noted below, the text of and illustrations in this documentation are licensed by Red Hat under the [Creative Commons Attribution–Share Alike 3.0 Unported license](#). If you distribute this document or an adaptation of it, you must provide the URL for the original version. Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

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

XFS is a trademark or registered trademark of Hewlett Packard Enterprise Development LP or its subsidiaries in the United States and other countries.

The OpenStack® Word Mark and OpenStack logo are trademarks or registered trademarks of the Linux Foundation, used under license.

All other trademarks are the property of their respective owners.

GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. link:<https://fsf.org/>. Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

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

Licensors 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



Copyright 2026. All rights reserved.

www.redhat.com