Red Hat Ansible Automation Platform 2.3

Red Hat Ansible Automation Platform Installation Guide

Learn how to install Red Hat Ansible Automation Platform based on supported installation scenarios.

Last Updated: 2024-04-18

Red Hat Ansible Automation Platform 2.3 Red Hat Ansible Automation Platform Installation Guide

Learn how to install Red Hat Ansible Automation Platform based on supported installation scenarios.

Legal Notice

Abstract

Providing Feedback: If you have a suggestion to improve this documentation, or find an error, please contact technical support at to create an issue on the Ansible Automation Platform Jira project using the Docs component.

Table of Contents

PREFACE

Thank you for your interest in Red Hat Ansible Automation Platform. Ansible Automation Platform is a commercial offering that helps teams manage complex multi-tier deployments by adding control, knowledge, and delegation to Ansible-powered environments.

This guide helps you to understand the installation requirements and processes behind installing Ansible Automation Platform. This document has been updated to include information for the latest release of Ansible Automation Platform.

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message.

CHAPTER 1. RED HAT ANSIBLE AUTOMATION PLATFORM INSTALLATION OVERVIEW

The Red Hat Ansible Automation Platform installation program offers you flexibility, allowing you to install Ansible Automation Platform using a number of supported installation scenarios.

Regardless of the installation scenario you choose, installing Ansible Automation Platform involves the following steps:

Editing the Red Hat Ansible Automation Platform installer inventory file

The Ansible Automation Platform installer inventory file allows you to specify your installation scenario and describe host deployments to Ansible. The examples provided in this document show the parameter specifications needed to install that scenario for your deployment.

Running the Red Hat Ansible Automation Platform installer setup script

The setup script installs your Private Automation Hub using the required parameters defined in the inventory file.

Verifying automation controller installation

After installing Ansible Automation Platform, you can verify that the installation has been successful by logging in to the automation controller.

Verifying automation hub installation

After installing Ansible Automation Platform, you can verify that the installation has been successful by logging in to the automation hub.

Post-installation steps

After successful installation, you can begin using the features of Ansible Automation Platform.

Additional resources

For more information about the supported installation scenarios, see the Red Hat Ansible Automation Platform Planning Guide.

1.1. PREREQUISITES

- You chose and obtained a platform installer from the Red Hat Ansible Automation Platform Product Software.
- You are installing on a machine that meets base system requirements.
- You have updated all of the packages to the recent version of your RHEL nodes.

WARNING

You may experience errors if you do not fully upgrade your RHEL nodes prior to your Ansible Automation Platform installation.

- You have created a Red Hat Registry Service Account, using the instructions in the Creating Registry Service Accounts guide.

Additional resources

For more information about obtaining a platform installer or system requirements, refer to the Red Hat Ansible Automation Platform system requirements in the *Red Hat Ansible Automation Platform Planning Guide*.

CHAPTER 2. SYSTEM REQUIREMENTS

Use this information when planning your Red Hat Ansible Automation Platform installations and designing automation mesh topologies that fit your use case.

Prerequisites

- You must be able to obtain root access either through the **sudo** command, or through privilege escalation. For more on privilege escalation see Understanding Privilege Escalation.
- You must be able to de-escalate privileges from root to users such as: AWX, PostgreSQL, or Pulp.
- You must configure an NTP client on all nodes. For more information, see Configuring NTP server using Chrony.

2.1. RED HAT ANSIBLE AUTOMATION PLATFORM SYSTEM REQUIREMENTS

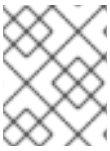
Your system must meet the following minimum system requirements to install and run Red Hat Ansible Automation Platform.

Table 2.1. Base system

Requirement	Required	Notes
Subscription	Valid Red Hat Ansible Automation Platform	
OS	Red Hat Enterprise Linux 8.4 or later 64-bit (x86)	Red Hat Ansible Automation Platform is also supported on OpenShift, see Deploying the Red Hat Ansible Automation Platform operator on OpenShift Container Platform for more information.
Ansible	version 2.14 (to install)	Ansible Automation Platform ships with execution environments that contain ansible-core 2.14.
Python	3.8 or later	
Browser	A currently supported version of Mozilla FireFox or Google Chrome	
Database	PostgreSQL version 13	

The following are necessary for you to work with project updates and collections:

- Ensure that the following domain names are part of either the firewall or the proxy's allowlist for successful connection and download of collections from automation hub or Galaxy server:
  - **galaxy.ansible.com**
  - **cloud.redhat.com**
  - **console.redhat.com**
  - **sso.redhat.com**
- SSL inspection must be disabled either when using self signed certificates or for the Red Hat domains.



NOTE

The requirements for systems managed by Ansible Automation Platform are the same as for Ansible. See Getting Started in the Ansible *User Guide*.

Additional notes for Red Hat Ansible Automation Platform requirements

- The requirements for systems managed by Ansible Automation Platform are the same as for Ansible. See Getting Started in the Ansible *User Guide*.
- Although Red Hat Ansible Automation Platform depends on Ansible Playbooks and requires the installation of the latest stable version of Ansible before installing automation controller, manual installations of Ansible are no longer required.
- For new installations, automation controller installs the latest release package of Ansible 2.3.
- If performing a bundled Ansible Automation Platform installation, the installation program attempts to install Ansible (and its dependencies) from the bundle for you.
- If you choose to install Ansible on your own, the Ansible Automation Platform installation program detects that Ansible has been installed and does not attempt to reinstall it.

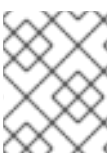
NOTE

You must install Ansible using a package manager such as **yum**, and the latest stable version of the package manager must be installed for Red Hat Ansible Automation Platform to work properly. Ansible version 2.14 is required for versions 2.3 and later.

2.2. AUTOMATION CONTROLLER SYSTEM REQUIREMENTS

Automation controller is a distributed system, where different software components can be co-located or deployed across multiple compute nodes. In the installer, node types of control, hybrid, execution, and hop are provided as abstractions to help you design the topology appropriate for your use case.

Use the following recommendations for node sizing:



NOTE

On control and hybrid nodes, allocate a minimum of 20 GB to **/var/lib/awx** for execution environment storage.

Execution nodes

Runs automation. Increases memory and CPU to increase capacity for running more forks

Requirement	Required
RAM	16 GB
CPUs	4
Local disk	40GB minimum

Control nodes

Processes events and runs cluster jobs including project updates and cleanup jobs. Increasing CPU and memory can help with job event processing.

Requirement	Required
RAM	16 GB
CPUs	4
Local disk	• 40GB minimum with at least 20GB available under /var/lib/awx • Storage volume must be rated for a minimum baseline of 1500 IOPS • Projects are stored on control and hybrid nodes, and for the duration of jobs, are also stored on execution nodes. If the cluster has many large projects, consider doubling the GB in /var/lib/awx/projects, to avoid disk space errors

Hybrid nodes

Runs both automation and cluster jobs. Comments on CPU and memory for execution and control nodes also apply to this node type.

Requirement	Required
RAM	16 GB
CPUs	4

Requirement	Required
Local disk	• 40GB minimum with at least 20GB available under /var/lib/awx • Storage volume must be rated for a minimum baseline of 1500 IOPS • Projects are stored on control and hybrid nodes, and for the duration of jobs, are also stored on execution nodes. If the cluster has many large projects, consider doubling the GB in /var/lib/awx/projects, to avoid disk space errors

Hop nodes

Serves to route traffic from one part of the automation mesh to another (for example, could be a bastion host into another network). RAM could affect throughput, CPU activity is low. Network bandwidth and latency are generally a more important factor than either RAM or CPU.

Requirement	Required
RAM	16 GB
CPUs	4
Local disk	40GB

- Actual RAM requirements vary based on how many hosts automation controller will manage simultaneously (which is controlled by the **forks** parameter in the job template or the system **ansible.cfg** file). To avoid possible resource conflicts, Ansible recommends 1 GB of memory per 10 forks + 2 GB reservation for automation controller, see Automation controller Capacity Determination and Job Impact for further details. If **forks** is set to 400, 42 GB of memory is recommended.
- Automation controller hosts check if **umask** is set to 0022. If not, the setup fails. Set **umask=0022** to avoid this error.
- A larger number of hosts can be addressed, but if the fork number is less than the total host count, more passes across the hosts are required. You can avoid these RAM limitations by using any of the following approaches:
  - Use rolling updates.
  - Use the provisioning callback system built into automation controller, where each system requesting configuration enters a queue and is processed as quickly as possible.
  - In cases where automation controller is producing or deploying images such as AMIs.

Additional resources

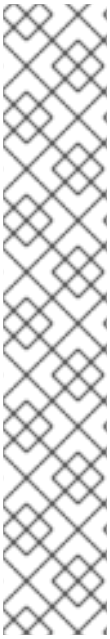
- For more information about obtaining an automation controller subscription, see Import a subscription.
- For questions, contact Ansible support through the Red Hat Customer portal .

2.3. AUTOMATION HUB SYSTEM REQUIREMENTS

Automation hub enables you to discover and use new certified automation content from Red Hat Ansible and Certified Partners. On Ansible automation hub, you can discover and manage Ansible Collections, which are supported automation content developed by Red Hat and its partners for use cases such as cloud automation, network automation, and security automation.

Automation hub has the following system requirements:

Requirement	Required	Notes
RAM	8 GB minimum	• 8 GB RAM (minimum and recommended for Vagrant trial installations) • 8 GB RAM (minimum for external standalone PostgreSQL databases) • For capacity based on forks in your configuration, see additional resources
CPUs	2 minimum	For capacity based on forks in your configuration, see additional resources.
Local disk	60 GB disk	A minimum of 40GB should be dedicated to /var for collection storage.



NOTE

Private automation hub

If you install private automation hub from an internal address, and have a certificate which only encompasses the external address, this can result in an installation which cannot be used as container registry without certificate issues.

To avoid this, use the **automationhub_main_url** inventory variable with a value like https://pah.example.com linking to the private automation hub node in the installation inventory file.

This adds the external address to **/etc/pulp/settings.py**.

This implies that you only want to use the external address.

For information on inventory file variables, see Inventory File Variables in the *Red Hat Ansible Automation Platform Installation Guide*.

2.4. POSTGRESQL REQUIREMENTS

Red Hat Ansible Automation Platform uses PostgreSQL 13.

- PostgreSQL user passwords are hashed with SCRAM-SHA-256 secure hashing algorithm before storing in the database.
- To determine if your automation controller instance has access to the database, you can do so with the command, **awx-manage check_db**.

Table 2.2. Database

Service	Required	Notes
Each automation controller	40 GB dedicated hard disk space	• Dedicate a minimum of 20 GB to **/var/** for file and working directory storage. • Storage volume must be rated for a minimum baseline of 1500 IOPS. • Projects are stored on control and hybrid nodes, and for the duration of jobs, are also stored on execution nodes. If the cluster has many large projects, consider having twice the GB in /var/lib/awx/projects, to avoid disk space errors. • 150 GB+ recommended

Service	Required	Notes
Each automation hub	60 GB dedicated hard disk space	Storage volume must be rated for a minimum baseline of 1500 IOPS.
Database	20 GB dedicated hard disk space	• 150 GB+ recommended • Storage volume must be rated for a high baseline IOPS (1500 or more). • All automation controller data is stored in the database. Database storage increases with the number of hosts managed, number of jobs run, number of facts stored in the fact cache, and number of tasks in any individual job. For example, a playbook run every hour (24 times a day) across 250 hosts, with 20 tasks, will store over 800000 events in the database every week. • If not enough space is reserved in the database, old job runs and facts must be cleaned on a regular basis. Refer to Management Jobs in the *Automation Controller Administration Guide* for more information

PostgreSQL Configurations

Optionally, you can configure the PostgreSQL database as separate nodes that are not managed by the Red Hat Ansible Automation Platform installer. When the Ansible Automation Platform installer manages the database server, it configures the server with defaults that are generally recommended for most workloads. However, you can adjust these PostgreSQL settings for standalone database server node where **ansible_memtotal_mb** is the total memory size of the database server:

```
max_connections == 1024
shared_buffers == ansible_memtotal_mb*0.3
work_mem == ansible_memtotal_mb*0.03
maintenance_work_mem == ansible_memtotal_mb*0.04
```

Additional resources

For more detail on tuning your PostgreSQL server, see the PostgreSQL documentation.

2.4.1. Benchmarking storage performance for the Ansible Automation Platform PostgreSQL database

The following procedure describes how to benchmark the write/read IOPS performance of the storage system to check whether the minimum Ansible Automation Platform PostgreSQL database requirements are met.

Prerequisites

- You have installed the Flexible I/O Tester (fio) storage performance benchmarking tool.
  To install fio, run the following command as the root user:

```
# yum -y install fio
```

- You have adequate disk space to store the fio test data log files.
  The examples shown in the procedure require at least 60GB disk space in the **/tmp** directory:
  - **numjobs** sets the number of jobs run by the command.
  - **size=10G** sets the file size generated by each job.

  To reduce the amount of test data, adjust the value of the **size** parameter.

Procedure

1. Run a random write test:

```
$ fio --name=write_iops --directory=/tmp --numjobs=3 --size=10G \
--time_based --runtime=60s --ramp_time=2s --ioengine=libaio --direct=1 \
--verify=0 --bs=4K --iodepth=64 --rw=randwrite \
--group_reporting=1 > /tmp/fio_benchmark_write_iops.log \
2>> /tmp/fio_write_iops_error.log
```

2. Run a random read test:

```
$ fio --name=read_iops --directory=/tmp \
--numjobs=3 --size=10G --time_based --runtime=60s --ramp_time=2s \
--ioengine=libaio --direct=1 --verify=0 --bs=4K --iodepth=64 --rw=randread \
--group_reporting=1 > /tmp/fio_benchmark_read_iops.log \
2>> /tmp/fio_read_iops_error.log
```

3. Review the results:
   In the log files written by the benchmark commands, search for the line beginning with **iops**. This line shows the minimum, maximum, and average values for the test.

   The following example shows the line in the log file for the random read test:

```
$ cat /tmp/fio_benchmark_read_iops.log
read_iops: (g=0): rw=randread, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B,
ioengine=libaio, iodepth=64
[…]
  iops        : min=50879, max=61603, avg=56221.33, stdev=679.97, samples=360
[…]
```

You must review, monitor, and revisit the log files according to your own business requirements, application workloads, and new demands.

CHAPTER 3. INSTALLING RED HAT ANSIBLE AUTOMATION PLATFORM

Ansible Automation Platform is a modular platform and you can deploy automation controller with other automation platform components, such as automation hub. For more information about the components provided with Ansible Automation Platform, see Red Hat Ansible Automation Platform components in the Red Hat Ansible Automation Platform Planning Guide.

There are a number of supported installation scenarios for Red Hat Ansible Automation Platform. To install Red Hat Ansible Automation Platform, you must edit the inventory file parameters to specify your installation scenario using one of the following examples:

- Standalone automation controller with internal database
- Single automation controller with external (installer managed) database
- Single automation controller with external (customer provided) database
- Ansible Automation Platform with an external (installer managed) database
- Ansible Automation Platform with an external (customer provided) database
- Standalone automation hub with internal database
- Single automation hub with external (installer managed) database
- Single automation hub with external (customer provided) database
- LDAP configuration on private automation hub

3.1. EDITING THE RED HAT ANSIBLE AUTOMATION PLATFORM INSTALLER INVENTORY FILE

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

Procedure

1. Navigate to the installer:
   a. [RPM installed package]

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

   b. [bundled installer]

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

   c. [online installer]

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

2. Open the **inventory** file with a text editor.

3. Edit **inventory** file parameters to specify your installation scenario. Use one of the supported Installation scenario examples to update your **inventory** file.

Additional resources

For a comprehensive list of pre-defined variables used in Ansible installation inventory files, see Inventory file variables.

3.1.1. Inventory file examples based on installation scenarios

Red Hat supports several installation scenarios for Ansible Automation Platform. Review the following examples and select those suitable for your preferred installation scenario.

IMPORTANT

- For Red Hat Ansible Automation Platform or automation hub: Add an automation hub host in the **[automationhub]** group.
- For internal databases: **[database]** cannot be used to point to another host in the Ansible Automation Platform cluster. The database host set to be installed needs to be a unique host.
- Do not install automation controller and automation hub on the same node for versions of Ansible Automation Platform in a production or customer environment. This can cause contention issues and heavy resource use.
- Provide a reachable IP address or fully qualified domain name (FQDN) for the **[automationhub]** and **[automationcontroller]** hosts to ensure users can sync and install content from automation hub from a different node. Do not use 'localhost'.
- Do not use special characters for **pg_password**. It can cause the setup to fail.
- Enter your Red Hat Registry Service Account credentials in **registry_username** and **registry_password** to link to the Red Hat container registry.
- The inventory file variables **registry_username** and **registry_password** are only required if a non-bundle installer is used.

3.1.1.1. Standalone automation controller with internal database

Use this example to populate the inventory file to install Red Hat Ansible Automation Platform. This installation inventory file includes a single automation controller node with an internal database.

```
[automationcontroller]
controller.acme.org

[all:vars]
admin_password='<password>'
pg_host=''
pg_port='5432'
pg_database='awx'
pg_username='awx'
pg_password='<password>'
pg_sslmode='prefer'  # set to 'verify-full' for client-side enforced SSL
```

```
registry_url='registry.redhat.io'
registry_username='<registry username>'
registry_password='<registry password>'


# SSL-related variables
# If set, this will install a custom CA certificate to the system trust store.
# custom_ca_cert=/path/to/ca.crt
# Certificate and key to install in nginx for the web UI and API
# web_server_ssl_cert=/path/to/tower.cert
# web_server_ssl_key=/path/to/tower.key
# Server-side SSL settings for PostgreSQL (when we are installing it).
# postgres_use_ssl=False
# postgres_ssl_cert=/path/to/pgsql.crt
# postgres_ssl_key=/path/to/pgsql.key
```

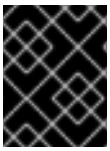
3.1.1.2. Single automation controller with external (installer managed) database

Use this example to populate the inventory file to install Red Hat Ansible Automation Platform. This installation inventory file includes a single automation controller node with an external database on a separate node.

```
[automationcontroller]
controller.acme.org

[database]
data.acme.org

[all:vars]
admin_password='<password>'
pg_host='data.acme.org'
pg_port='5432'
pg_database='awx'
pg_username='awx'
pg_password='<password>'
pg_sslmode='prefer'  # set to 'verify-full' for client-side enforced SSL

registry_url='registry.redhat.io'
registry_username='<registry username>'
registry_password='<registry password>'

# SSL-related variables
# If set, this will install a custom CA certificate to the system trust store.
# custom_ca_cert=/path/to/ca.crt
# Certificate and key to install in nginx for the web UI and API
# web_server_ssl_cert=/path/to/tower.cert
# web_server_ssl_key=/path/to/tower.key
# Server-side SSL settings for PostgreSQL (when we are installing it).
# postgres_use_ssl=False
# postgres_ssl_cert=/path/to/pgsql.crt
# postgres_ssl_key=/path/to/pgsql.key
```

3.1.1.3. Single automation controller with external (customer provided) database

Use this example to populate the inventory file to install Red Hat Ansible Automation Platform. This installation inventory file includes a single automation controller node with an external database on a separate node that is not managed by the platform installer.



IMPORTANT

This example does not have a host under the database group. This indicates to the installer that the database already exists, and is being managed elsewhere.

```
[automationcontroller]
controller.acme.org

[database]

[all:vars]
admin_password='<password>'

pg_host='data.acme.org'
pg_port='5432'
pg_database='awx'
pg_username='awx'
pg_password='<password>'
pg_sslmode='prefer'  # set to 'verify-full' for client-side enforced SSL

registry_url='registry.redhat.io'
registry_username='<registry username>'
registry_password='<registry password>'


# SSL-related variables
# If set, this will install a custom CA certificate to the system trust store.
# custom_ca_cert=/path/to/ca.crt
# Certificate and key to install in nginx for the web UI and API
# web_server_ssl_cert=/path/to/tower.cert
# web_server_ssl_key=/path/to/tower.key
# Server-side SSL settings for PostgreSQL (when we are installing it).
# postgres_use_ssl=False
# postgres_ssl_cert=/path/to/pgsql.crt
# postgres_ssl_key=/path/to/pgsql.key
```

3.1.1.4. Ansible Automation Platform with an external (installer managed) database

Use this example to populate the inventory file to install Ansible Automation Platform. This installation inventory file includes two automation controller nodes, two execution nodes, and automation hub with an external managed database.

```
# Automation Controller Nodes
# There are two valid node_types that can be assigned for this group.
# A node_type=control implies that the node will only be able to run
# project and inventory updates, but not regular jobs.
# A node_type=hybrid will have the ability to run everything.
# If you do not define the node_type, it defaults to hybrid.
#
# control.example node_type=control
```

```
# hybrid.example  node_type=hybrid
# hybrid2.example <- this will default to hybrid

[automationcontroller]
controller1.acme.org node_type=control
controller2.acme.org node_type=control

# Execution Nodes
# There are two valid node_types that can be assigned for this group.
# A node_type=hop implies that the node will forward jobs to an execution node.
# A node_type=execution implies that the node will be able to run jobs.
# If you do not define the node_type, it defaults to execution.
#
# hop.example node_type=hop
# execution.example  node_type=execution
# execution2.example <- this will default to execution

[execution_nodes]
execution1.acme.org node_type=execution
execution2.acme.org node_type=execution

[automationhub]
automationhub.acme.org

[database]
data.acme.org

[all:vars]
admin_password='<password>'
pg_host='data.acme.org'
pg_port='5432'
pg_database='awx'
pg_username='awx'
pg_password='<password>'
pg_sslmode='prefer'  # set to 'verify-full' for client-side enforced SSL

registry_url='registry.redhat.io'
registry_username='<registry username>'
registry_password='<registry password>'

# Receptor Configuration
#
receptor_listener_port=27199

# Automation Hub Configuration
#
automationhub_admin_password='<password>'
automationhub_pg_host='data.acme.org'
automationhub_pg_port='5432'
automationhub_pg_database='automationhub'
automationhub_pg_username='automationhub'
automationhub_pg_password='<password>'
automationhub_pg_sslmode='prefer'

# The default install will deploy a TLS enabled Automation Hub.
# If for some reason this is not the behavior wanted one can
```

```
# disable TLS enabled deployment.
#
# automationhub_disable_https = False
# The default install will generate self-signed certificates for the Automation
# Hub service. If you are providing valid certificate via automationhub_ssl_cert
# and automationhub_ssl_key, one should toggle that value to True.
#
# automationhub_ssl_validate_certs = False
# SSL-related variables
# If set, this will install a custom CA certificate to the system trust store.
# custom_ca_cert=/path/to/ca.crt
# Certificate and key to install in nginx for the web UI and API
# web_server_ssl_cert=/path/to/tower.cert
# web_server_ssl_key=/path/to/tower.key
# Certificate and key to install in Automation Hub node
# automationhub_ssl_cert=/path/to/automationhub.cert
# automationhub_ssl_key=/path/to/automationhub.key
# Server-side SSL settings for PostgreSQL (when we are installing it).
# postgres_use_ssl=False
# postgres_ssl_cert=/path/to/pgsql.crt
# postgres_ssl_key=/path/to/pgsql.key
```

3.1.1.5. Ansible Automation Platform with an external (customer provided) database

Use this example to populate the inventory file to install Red Hat Ansible Automation Platform. This installation inventory file includes one of each node type; control, hybrid, hop, and execution, and automation hub with an external managed database that is not managed by the platform installer.

IMPORTANT

This example does not have a host under the database group. This indicates to the installer that the database already exists, and is being managed elsewhere.

```
# Automation Controller Nodes
# There are two valid node_types that can be assigned for this group.
# A node_type=control implies that the node will only be able to run
# project and inventory updates, but not regular jobs.
# A node_type=hybrid will have the ability to run everything.
# If you do not define the node_type, it defaults to hybrid.
#
# control.example node_type=control
# hybrid.example  node_type=hybrid
# hybrid2.example <- this will default to hybrid

[automationcontroller]
hybrid1.acme.org node_type=hybrid
controller1.acme.org node_type=control

# Execution Nodes
# There are two valid node_types that can be assigned for this group.
# A node_type=hop implies that the node will forward jobs to an execution node.
# A node_type=execution implies that the node will be able to run jobs.
# If you do not define the node_type, it defaults to execution.
#
```

```
# hop.example node_type=hop
# execution.example  node_type=execution
# execution2.example <- this will default to execution

[execution_nodes]
hop1.acme.org node_type=hop
execution1.acme.org node_type=execution

[automationhub]
automationhub.acme.org

[database]

[all:vars]
admin_password='<password>'
pg_host='data.acme.org'
pg_port='5432'
pg_database='awx'
pg_username='awx'
pg_password='<password>'
pg_sslmode='prefer'  # set to 'verify-full' for client-side enforced SSL

registry_url='registry.redhat.io'
registry_username='<registry username>'
registry_password='<registry password>'

# Receptor Configuration
#
receptor_listener_port=27199

# Automation Hub Configuration
#
automationhub_admin_password='<password>'
automationhub_pg_host='data.acme.org'
automationhub_pg_port='5432'
automationhub_pg_database='automationhub'
automationhub_pg_username='automationhub'
automationhub_pg_password='<password>'
automationhub_pg_sslmode='prefer'

# The default install will deploy a TLS enabled Automation Hub.
# If for some reason this is not the behavior wanted one can
# disable TLS enabled deployment.
#
# automationhub_disable_https = False
# The default install will generate self-signed certificates for the Automation
# Hub service. If you are providing valid certificate via automationhub_ssl_cert
# and automationhub_ssl_key, one should toggle that value to True.
#
# automationhub_ssl_validate_certs = False
# SSL-related variables
# If set, this will install a custom CA certificate to the system trust store.
# custom_ca_cert=/path/to/ca.crt
# Certificate and key to install in nginx for the web UI and API
# web_server_ssl_cert=/path/to/tower.cert
# web_server_ssl_key=/path/to/tower.key
```

```
# Certificate and key to install in Automation Hub node
# automationhub_ssl_cert=/path/to/automationhub.cert
# automationhub_ssl_key=/path/to/automationhub.key
# Server-side SSL settings for PostgreSQL (when we are installing it).
# postgres_use_ssl=False
# postgres_ssl_cert=/path/to/pgsql.crt
# postgres_ssl_key=/path/to/pgsql.key
```

3.1.1.6. Standalone automation hub with internal database

Use this example to populate the inventory file to deploy a standalone instance of automation hub with an internal database.

```
[automationcontroller]


[automationhub]
automationhub.acme.org ansible_connection=local

[all:vars]
registry_url='registry.redhat.io'
registry_username='<registry username>'
registry_password='<registry password>'

automationhub_admin_password= <PASSWORD>

automationhub_pg_host=''
automationhub_pg_port='5432'

automationhub_pg_database='automationhub'
automationhub_pg_username='automationhub'
automationhub_pg_password=<PASSWORD>
automationhub_pg_sslmode='prefer'

# The default install will deploy a TLS enabled Automation Hub.
# If for some reason this is not the behavior wanted one can
# disable TLS enabled deployment.
#
# automationhub_disable_https = False
# The default install will generate self-signed certificates for the Automation
# Hub service. If you are providing valid certificate via automationhub_ssl_cert
# and automationhub_ssl_key, one should toggle that value to True.
#
# automationhub_ssl_validate_certs = False
# SSL-related variables
# If set, this will install a custom CA certificate to the system trust store.
# custom_ca_cert=/path/to/ca.crt
# Certificate and key to install in Automation Hub node
# automationhub_ssl_cert=/path/to/automationhub.cert
# automationhub_ssl_key=/path/to/automationhub.key
```

3.1.1.7. Single automation hub with external (installer managed) database

Use this example to populate the inventory file to deploy a single instance of automation hub with an external (installer managed) database.

```
[automationcontroller]

[automationhub]
automationhub.acme.org

[database]
data.acme.org

[all:vars]
registry_url='registry.redhat.io'
registry_username='<registry username>'
registry_password='<registry password>'

automationhub_admin_password= <PASSWORD>

automationhub_pg_host='data.acme.org'
automationhub_pg_port='5432'

automationhub_pg_database='automationhub'
automationhub_pg_username='automationhub'
automationhub_pg_password=<PASSWORD>
automationhub_pg_sslmode='prefer'

# The default install will deploy a TLS enabled Automation Hub.
# If for some reason this is not the behavior wanted one can
# disable TLS enabled deployment.
#
# automationhub_disable_https = False
# The default install will generate self-signed certificates for the Automation
# Hub service. If you are providing valid certificate via automationhub_ssl_cert
# and automationhub_ssl_key, one should toggle that value to True.
#
# automationhub_ssl_validate_certs = False
# SSL-related variables
# If set, this will install a custom CA certificate to the system trust store.
# custom_ca_cert=/path/to/ca.crt
# Certificate and key to install in Automation Hub node
# automationhub_ssl_cert=/path/to/automationhub.cert
# automationhub_ssl_key=/path/to/automationhub.key
```

3.1.1.8. Single automation hub with external (customer provided) database

Use this example to populate the inventory file to deploy a single instance of automation hub with an external database that is not managed by the platform installer.

IMPORTANT

This example does not have a host under the database group. This indicates to the installer that the database already exists, and is being managed elsewhere.

```
[automationcontroller]

[automationhub]
automationhub.acme.org
```

```
[database]

[all:vars]
registry_url='registry.redhat.io'
registry_username='<registry username>'
registry_password='<registry password>'

automationhub_admin_password= <PASSWORD>

automationhub_pg_host='data.acme.org'
automationhub_pg_port='5432'

automationhub_pg_database='automationhub'
automationhub_pg_username='automationhub'
automationhub_pg_password=<PASSWORD>
automationhub_pg_sslmode='prefer'

# The default install will deploy a TLS enabled Automation Hub.
# If for some reason this is not the behavior wanted one can
# disable TLS enabled deployment.
#
# automationhub_disable_https = False
# The default install will generate self-signed certificates for the Automation
# Hub service. If you are providing valid certificate via automationhub_ssl_cert
# and automationhub_ssl_key, one should toggle that value to True.
#
# automationhub_ssl_validate_certs = False
# SSL-related variables
# If set, this will install a custom CA certificate to the system trust store.
# custom_ca_cert=/path/to/ca.crt
# Certificate and key to install in Automation Hub node
# automationhub_ssl_cert=/path/to/automationhub.cert
# automationhub_ssl_key=/path/to/automationhub.key
```

3.1.1.9. LDAP configuration on private automation hub

You must set the following six variables in your Red Hat Ansible Automation Platform installer inventory file to configure your private automation hub for LDAP authentication:

- **automationhub_authentication_backend**
- **automationhub_ldap_server_uri**
- **automationhub_ldap_bind_dn**
- **automationhub_ldap_bind_password**
- **automationhub_ldap_user_search_base_dn**
- **automationhub_ldap_group_search_base_dn**

If any of these variables are missing, the Ansible Automation installer will not complete the installation.

3.1.1.9.1. Setting up your inventory file variables

When you configure your private automation hub with LDAP authentication, you must set the proper variables in your inventory files during the installation process.

Procedure

1. Access your inventory file according to the procedure in Editing the Red Hat Ansible Automation Platform installer inventory file.
2. Use the following example as a guide to set up your Ansible Automation Platform inventory file:

```
automationhub_authentication_backend = "ldap"

automationhub_ldap_server_uri = "ldap://ldap:389"   (for LDAPs use
automationhub_ldap_server_uri = "ldaps://ldap-server-fqdn")
automationhub_ldap_bind_dn = "cn=admin,dc=ansible,dc=com"
automationhub_ldap_bind_password = "GoodNewsEveryone"
automationhub_ldap_user_search_base_dn = "ou=people,dc=ansible,dc=com"
automationhub_ldap_group_search_base_dn = "ou=people,dc=ansible,dc=com"
```

NOTE

The following variables will be set with default values, unless you set them with other options.

```
auth_ldap_user_search_scope= 'SUBTREE'
auth_ldap_user_search_filter= '(uid=%(user)s)'
auth_ldap_group_search_scope= 'SUBTREE'
auth_ldap_group_search_filter= '(objectClass=Group)'
auth_ldap_group_type_class= 'django_auth_ldap.config:GroupOfNamesType'
```

3. Optional: Set up extra parameters in your private automation hub such as user groups, superuser access, or mirroring. Go to Configuring extra LDAP parameters to complete this optional step.

3.1.1.9.2. Configuring extra LDAP parameters

If you plan to set up superuser access, user groups, mirroring or other extra parameters, you can create a YAML file that comprises them in your **ldap_extra_settings** dictionary.

Procedure

1. Create a YAML file that contains **ldap_extra_settings**.
   - Example:

```
#ldapextras.yml
---
ldap_extra_settings:
 <LDAP_parameter>: <Values>

...
```

2. Add any parameters that you require for your setup. The following examples describe the LDAP parameters that you can set in **ldap_extra_settings**:

   - Use this example to set up a superuser flag based on membership in an LDAP group.

     ```
     #ldapextras.yml
     ---
     ldap_extra_settings:
       AUTH_LDAP_USER_FLAGS_BY_GROUP: {"is_superuser": "cn=pah-
     admins,ou=groups,dc=example,dc=com",}
     ...
     ```

   - Use this example to set up superuser access.

     ```
     #ldapextras.yml
     ---
     ldap_extra_settings:
       AUTH_LDAP_USER_FLAGS_BY_GROUP: {"is_superuser": "cn=pah-
     admins,ou=groups,dc=example,dc=com",}
     ...
     ```

   - Use this example to mirror all LDAP groups you belong to.

     ```
     #ldapextras.yml
     ---
     ldap_extra_settings:
       AUTH_LDAP_MIRROR_GROUPS: True
     ...
     ```

   - Use this example to map LDAP user attributes (such as first name, last name, and email address of the user).

     ```
     #ldapextras.yml
     ---
     ldap_extra_settings:
       AUTH_LDAP_USER_ATTR_MAP: {"first_name": "givenName", "last_name": "sn",
     "email": "mail",}
     ...
     ```

   - Use the following examples to grant or deny access based on LDAP group membership:

     - To grant private automation hub access (for example, members of the **cn=pah-nosoupforyou,ou=groups,dc=example,dc=com** group):

       ```
       #ldapextras.yml
       ---
       ldap_extra_settings:
         AUTH_LDAP_REQUIRE_GROUP: 'cn=pah-
       nosoupforyou,ou=groups,dc=example,dc=com'
       ...
       ```

     - To deny private automation hub access (for example, members of the **cn=pah-nosoupforyou,ou=groups,dc=example,dc=com** group):

       ```
       #ldapextras.yml
       ```

```
---
ldap_extra_settings:
  AUTH_LDAP_REQUIRE_GROUP: 'cn=pah-
nosoupforyou,ou=groups,dc=example,dc=com'
...
```

- Use this example to enable LDAP debug logging.

```
#ldapextras.yml
---
ldap_extra_settings:
  GALAXY_LDAP_LOGGING: True
...
```

NOTE

If it is not practical to re-run **setup.sh** or if debug logging is enabled for a short time, you can add a line containing **GALAXY_LDAP_LOGGING: True** manually to the **/etc/pulp/settings.py** file on private automation hub. Restart both **pulpcore-api.service** and **nginx.service** for the changes to take effect. To avoid failures due to human error, use this method only when necessary.

- Use this example to configure LDAP caching by setting the variable **AUTH_LDAP_CACHE_TIMEOUT**.

```
#ldapextras.yml
---
ldap_extra_settings:
  AUTH_LDAP_CACHE_TIMEOUT: 3600
...
```

3. Run **setup.sh -e @ldapextras.yml** during private automation hub installation. .Verification To verify you have set up correctly, confirm you can view all of your settings in the **/etc/pulp/settings.py** file on your private automation hub.

3.2. RUNNING THE RED HAT ANSIBLE AUTOMATION PLATFORM INSTALLER SETUP SCRIPT

After you update the inventory file with required parameters for installing your private automation hub, run the installer setup script.

Procedure

- Run the **setup.sh** script

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

Installation of Red Hat Ansible Automation Platform will begin.

3.3. VERIFYING INSTALLATION OF AUTOMATION CONTROLLER

Verify that you installed automation controller successfully by logging in with the admin credentials you inserted in the **inventory** file.

Procedure

1. Navigate to the IP address specified for the automation controller node in the **inventory** file.
2. Log in with the Admin credentials you set in the **inventory** file.

NOTE

The automation controller server is accessible from port 80 (**https://<CONTROLLER_SERVER_NAME>/**) but redirects to port 443, so port 443 must also be available.

IMPORTANT

If the installation fails and you are a customer who has purchased a valid license for Red Hat Ansible Automation Platform, contact Ansible through the Red Hat Customer portal .

After a successful login to automation controller, your installation of Red Hat Ansible Automation Platform 2.3 is complete.

3.3.1. Additional automation controller configuration and resources

See the following resources to explore additional automation controller configurations.

Table 3.1. Resources to configure automation controller

Resource link	Description
Automation Controller Quick Setup Guide	Set up automation controller and run your first playbook
Automation Controller Administration Guide	Configure automation controller administration through customer scripts, management jobs, etc.
Configuring proxy support for Red Hat Ansible Automation Platform	Set up automation controller with a proxy server
Managing usability analytics and data collection from automation controller	Manage what automation controller information you share with Red Hat
Automation Controller User Guide	Review automation controller functionality in more detail

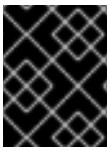
3.4. VERIFYING INSTALLATION OF AUTOMATION HUB

Verify that you installed your automation hub successfully by logging in with the admin credentials you inserted into the **inventory** file.

Procedure

Procedure

1. Navigate to the IP address specified for the automation hub node in the **inventory** file.
2. Log in with the Admin credentials you set in the **inventory** file.



IMPORTANT

If the installation fails and you are a customer who has purchased a valid license for Red Hat Ansible Automation Platform, contact Ansible through the Red Hat Customer portal .

After a successful login to automation hub, your installation of Red Hat Ansible Automation Platform 2.3 is complete.

3.4.1. Additional automation hub configuration and resources

See the following resources to explore additional automation hub configurations.

Table 3.2. Resources to configure automation controller

Resource link	Description
Managing user access in private automation hub	Configure user access for automation hub
Managing Red Hat Certified and Ansible Galaxy collections in automation hub	Add content to your automation hub
Publishing proprietary content collections in automation hub	Publish internally developed collections on your automation hub

3.5. POST-INSTALLATION STEPS

Whether you are a new Ansible Automation Platform user looking to start automating, or an existing administrator looking to migrate old Ansible content to your latest installed version of Red Hat Ansible Automation Platform, explore the next steps to begin leveraging the new features of Ansible Automation Platform 2.3:

3.5.1. Migrating data to Ansible Automation Platform 2.3

For platform administrators looking to complete an upgrade to the Ansible Automation Platform 2.3, there may be additional steps needed to migrate data to a new instance:

3.5.1.1. Migrating from legacy virtual environments (venvs) to automation execution environments

Ansible Automation Platform 2.3 moves you away from custom Python virtual environments (venvs) in favor of automation execution environments - containerized images that packages the necessary components needed to execute and scale your Ansible automation. This includes Ansible Core, Ansible Content Collections, Python dependencies, Red Hat Enterprise Linux UBI 8, and any additional package dependencies.

If you are looking to migrate your venvs to execution environments, you will (1) need to use the **awx-manage** command to list and export a list of venvs from your original instance, then (2) use **ansible-builder** to create execution environments.

Additional resources

- Upgrading to Automation Execution Environments guide
- Creating and Consuming Execution Environments .

3.5.1.2. Migrating to Ansible Engine 2.9 images using Ansible Builder

To migrate Ansible Engine 2.9 images for use with Ansible Automation Platform 2.3, the **ansible-builder** tool automates the process of rebuilding images (including its custom plugins and dependencies) for use with automation execution environments.

Additional resources

For more information on using Ansible Builder to build execution environments, see the Creating and Consuming Execution Environments.

3.5.1.3. Migrating to Ansible Core 2.13

When upgrading to Ansible Core 2.13, you need to update your playbooks, plugins, or other parts of your Ansible infrastructure in order to be supported by the latest version of Ansible Core. For instructions on updating your Ansible content for Ansible Core 2.13 compatibility, see the Ansible-core 2.13 Porting Guide.

3.5.2. Updating execution environment image locations

If your private automation hub was installed separately, you can update your execution environment image locations to point to your private automation hub. Use this procedure to update your execution environment image locations.

Procedure

1. Navigate to the directory containing **setup.sh**
2. Create **./group_vars/automationcontroller** by running the following command:

```
touch ./group_vars/automationcontroller
```

3. Paste the following content into **./group_vars/automationcontroller**, being sure to adjust the settings to fit your environment:

```
# Automation Hub Registry
registry_username: 'your-automation-hub-user'
registry_password: 'your-automation-hub-password'
registry_url: 'automationhub.example.org'
registry_verify_ssl: False

## Execution Environments
control_plane_execution_environment: 'automationhub.example.org/ee-supported-rhel8:latest'
```

```
global_job_execution_environments:
  - name: "Default execution environment"
    image: "automationhub.example.org/ee-supported-rhel8:latest"
  - name: "Ansible Engine 2.9 execution environment"
    image: "automationhub.example.org/ee-29-rhel8:latest"
  - name: "Minimal execution environment"
    image: "automationhub.example.org/ee-minimal-rhel8:latest"
```

4. Run the **./setup.sh** script

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

Verification

1. Log into Ansible Automation Platform as a user with system administrator access.
2. Navigate to **Administration → Execution Environments**.
3. In the Image column, confirm that the execution environment image location has changed from the default value of **<registry url>/ansible-automation-platform-<version>/<image name>:<tag>** to **<automation hub url>/<image name>:<tag>**.

3.5.3. Scale up your automation using automation mesh

The automation mesh component of the Red Hat Ansible Automation Platform simplifies the process of distributing automation across multi-site deployments. For enterprises with multiple isolated IT environments, automation mesh provides a consistent and reliable way to deploy and scale up automation across your execution nodes using a peer-to-peer mesh communication network.

When upgrading from version 1.x to the latest version of the Ansible Automation Platform, you will need to migrate the data from your legacy isolated nodes into execution nodes necessary for automation mesh. You can implement automation mesh by planning out a network of hybrid and control nodes, then editing the inventory file found in the Ansible Automation Platform installer to assign mesh-related values to each of your execution nodes.

For instructions on how to migrate from isolated nodes to execution nodes, see the Red Hat Ansible Automation Platform Upgrade and Migration Guide.

For information about automation mesh and the various ways to design your automation mesh for your environment, see the Red Hat Ansible Automation Platform automation mesh guide .

CHAPTER 4. DISCONNECTED INSTALLATION

4.1. ANSIBLE AUTOMATION PLATFORM INSTALLATION ON DISCONNECTED RHEL

Install Ansible Automation Platform automation controller and a private automation hub, with an installer-managed database located on the automation controller without an Internet connection.

4.1.1. Prerequisites

To install Ansible Automation Platform on a disconnected network, complete the following prerequisites:

- Create a subscription manifest.
- Download the Ansible Automation Platform setup bundle.
- Create DNS records for automation controller and private automation hub servers.

NOTE

The setup bundle includes additional components that make installing Ansible Automation Platform easier in a disconnected environment. These include the Ansible Automation Platform RPMs and the default execution environment (EE) images.

4.1.2. System Requirements

Hardware requirements are documented in the Automation Platform Installation Guide. Reference the "Red Hat Ansible Automation Platform Installation Guide" in the Ansible Automation Platform Product Documentation for your version of Ansible Automation Platform.

4.1.3. RPM Source

RPM dependencies for Ansible Automation Platform that come from the BaseOS and AppStream repositories are not included in the setup bundle. To add these dependencies, you must obtain access to BaseOS and AppStream repositories.

- Satellite is the recommended method from Red Hat to synchronize repositories
- reposync - Makes full copies of the required RPM repositories and hosts them on the disconnected network
- RHEL Binary DVD - Use the RPMs available on the RHEL 8 Binary DVD

NOTE

The RHEL Binary DVD method requires the DVD for supported versions of RHEL 8.4 or higher. See Red Hat Enterprise Linux Life Cycle for information on which versions of RHEL are currently supported.

4.2. SYNCHRONIZING RPM REPOSITORIES BY USING REPOSYNC

To perform a reposync you need a RHEL host that has access to the Internet. After the repositories are synced, you can move the repositories to the disconnected network hosted from a web server.

Procedure

1. Attach the BaseOS and AppStream required repositories:

```
# subscription-manager repos \
    --enable rhel-8-for-x86_64-baseos-rpms \
    --enable rhel-8-for-x86_64-appstream-rpms
```

2. Perform the reposync:

```
# dnf install yum-utils
# reposync -m --download-metadata --gpgcheck \
    -p /path/to/download
```

   a. Make certain that you use reposync with **--download-metadata** and without **--newest-only**. See [RHEL 8] Reposync.

   b. If not using **--newest-only** the repos downloaded will be ~90GB.

   c. If using **--newest-only** the repos downloaded will be ~14GB.

3. If you plan to use Red Hat Single Sign-On (RHSSO) you must also sync these repositories.

   a. jb-eap-7.3-for-rhel-8-x86_64-rpms

   b. rh-sso-7.4-for-rhel-8-x86_64-rpms

4. After the reposync is completed your repositories are ready to use with a web server.

5. Move the repositories to your disconnected network.

4.3. CREATING A NEW WEB SERVER TO HOST REPOSITORIES

If you do not have an existing web server to host your repositories, create one with the synced repositories.

Procedure

Use the following steps if creating a new web server.

1. Install prerequisites:

```
$ sudo dnf install httpd
```

2. Configure httpd to serve the repo directory:

```
/etc/httpd/conf.d/repository.conf

DocumentRoot '/path/to/repos'

<LocationMatch "^/+$">
    Options -Indexes
```

```
    ErrorDocument 403 /.noindex.html
</LocationMatch>

<Directory '/path/to/repos'>
    Options All Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>
```

3. Ensure that the directory is readable by an apache user:

```
$ sudo chown -R apache /path/to/repos
```

4. Configure SELinux:

```
$ sudo semanage fcontext -a -t httpd_sys_content_t "/path/to/repos(/.*)?"
$ sudo restorecon -ir /path/to/repos
```

5. Enable httpd:

```
$ sudo systemctl enable --now httpd.service
```

6. Open firewall:

```
$ sudo firewall-cmd --zone=public --add-service=http –add-service=https
--permanent
$ sudo firewall-cmd --reload
```

7. On automation controller and automation hub, add a repo file at */etc/yum.repos.d/local.repo*, add the optional repos if needed:

```
[Local-BaseOS]
name=Local BaseOS
baseurl=http://<webserver_fqdn>/rhel-8-for-x86_64-baseos-rpms
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release

[Local-AppStream]
name=Local AppStream
baseurl=http://<webserver_fqdn>/rhel-8-for-x86_64-appstream-rpms
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
```

4.4. ACCESSING RPM REPOSITORIES FOR LOCALLY MOUNTED DVD

If you are going to access the repositories from the DVD, it is necessary to set up a local repository. This section shows how to do that.

Procedure

1. Mount DVD or ISO

a. DVD

```
# mkdir /media/rheldvd && mount /dev/sr0 /media/rheldvd
```

b. ISO

```
# mkdir /media/rheldvd && mount -o loop rhrhel-8.6-x86_64-dvd.iso /media/rheldvd
```

2. Create yum repo file at **/etc/yum.repos.d/dvd.repo**

```
[dvd-BaseOS]
name=DVD for RHEL - BaseOS
baseurl=file:///media/rheldvd/BaseOS
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release

[dvd-AppStream]
name=DVD for RHEL - AppStream
baseurl=file:///media/rheldvd/AppStream
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
```

3. Import the gpg key

```
# rpm --import /media/rheldvd/RPM-GPG-KEY-redhat-release
```

NOTE

If the key is not imported you will see an error similar to

```
# Curl error (6): Couldn't resolve host name for
https://www.redhat.com/security/data/fd431d51.txt [Could not resolve host:
www.redhat.com]
```

In order to set up a repository see Need to set up yum repository for locally-mounted DVD on Red Hat Enterprise Linux 8.

4.5. ADDING A SUBSCRIPTION MANIFEST TO ANSIBLE AUTOMATION PLATFORM WITHOUT AN INTERNET CONNECTION

To add a subscription to Ansible Automation Platform without an Internet connection, create and import a subscription manifest.

Procedure

1. Login to **access.redhat.com**.
2. Navigate to **Subscriptions → Subscriptions**.
3. Click **Subscription Allocations**.

4. Click **Create New subscription allocation**.
5. Name the new subscription allocation.
6. Select **Satellite 6.14 → Satellite 6.14** as the type.
7. Click **Create**. The Details tab will open for your subscription allocation.
8. Click **Subscriptions** tab.
9. Click **Add Subscription**.
10. Find your Ansible Automation Platform subscription, in the Entitlements box **add** the number of entitlements you want to assign to your environment. A single entitlement is needed for each node that is managed by Ansible Automation Platform: server, network device, etc.
11. Click **Submit**.
12. Click **Export Manifest**.
13. This downloads a file *manifest_<allocation name>_<date>.zip* that be imported with automation controller after installation.

4.6. INSTALLING THE ANSIBLE AUTOMATION PLATFORM SETUP BUNDLE

The "bundle" version is strongly recommended for disconnected installations as it comes with the RPM content for Ansible Automation Platform as well as the default execution environment images that are uploaded to your private automation hub during the installation process.

4.6.1. Downloading the Setup Bundle

Procedure

1. Download the Ansible Automation Platform setup bundle package by navigating to https://access.redhat.com/downloads/content/480 and click **Download Now** for the Ansible Automation Platform 2.3 Setup Bundle.

4.6.1.1. Installing the Setup Bundle

The download and installation of the setup bundle needs to be located on automation controller. From automation controller, untar the bundle, edit the inventory file, and run the setup.

1. Untar the bundle

```
$ tar xvf \
  ansible-automation-platform-setup-bundle-2.3-1.2.tar.gz
$ cd ansible-automation-platform-setup-bundle-2.3-1.2
```

2. Edit the inventory file to include the required options
   a. automationcontroller group
   b. automationhub group

c. admin_password

d. pg_password

e. automationhub_admin_password

f. automationhub_pg_host, automationhub_pg_port

g. automationhub_pg_password

Example Inventory

```
[automationcontroller]
automationcontroller.example.org ansible_connection=local

[automationcontroller:vars]
peers=execution_nodes

[automationhub]
automationhub.example.org

[all:vars]
admin_password='password123'

pg_database='awx'
pg_username='awx'
pg_password='dbpassword123'

receptor_listener_port=27199

automationhub_admin_password='hubpassword123'

automationhub_pg_host='automationcontroller.example.org'
automationhub_pg_port='5432'

automationhub_pg_database='automationhub'
automationhub_pg_username='automationhub'
automationhub_pg_password='dbpassword123'
automationhub_pg_sslmode='prefer'
```

NOTE

The inventory should be kept intact after installation since it is used for backup, restore, and upgrade functions. Consider keeping a backup copy in a secure location, given that the inventory file contains passwords.

3. Run the AAP setup bundle executable as the root user

```
$ sudo -i
# cd /path/to/ansible-automation-platform-setup-bundle-2.3-1.2
# ./setup.sh
```

4. Once installation is complete, navigate to the Fully Qualified Domain Name (FQDN) for the automation controller node that was specified in the installation inventory file.

5. Log in with the administrator credentials specified in the installation inventory file.

4.7. COMPLETING POST INSTALLATION TASKS

4.7.1. Adding an automation controller Subscription

Procedure

1. Navigate to the FQDN of the Automation controller. Login with admin and the password you specified as **admin_password** in your inventory file.
2. Click **Browse** and select the *manifest.zip* you created earlier.
3. Click **Next**.
4. Uncheck **User analytics** and **Automation analytics**. These rely on an Internet connection and should be turned off.
5. Click **Next**.
6. Read the End User License Agreement and click **Submit** if you agree.

4.7.2. Updating the CA trust store

4.7.2.1. Self-Signed Certificates

By default, automation hub and automation controller are installed using self signed certificates. This creates an issue where automation controller does not trust automation hub's certificate and does not download the execution environments from automation hub. The solution is to import automation hub's CA cert as a trusted cert on automation controller. You can use SCP or directly copy and paste from one file into another to perform this action. The following steps are copied from a KB article found at https://access.redhat.com/solutions/6707451.

4.7.2.2. Copying the root certificate on the private automation hub to the automation controller using secure copy (SCP)

If SSH is available as the root user between automation controller and private automation hub, use SCP to copy the root certificate on private automation hub to automation controller and run **update-ca-trust** on automation controller to update the CA trust store.

On the Automation controller

```
$ sudo -i
# scp <hub_fqdn>:/etc/pulp/certs/root.crt
/etc/pki/ca-trust/source/anchors/automationhub-root.crt
# update-ca-trust
```

4.7.2.3. Copying and Pasting

If SSH is unavailable as root between private automation hub and automation controller, copy the contents of the file */etc/pulp/certs/root.crt* on private automation hub and paste it into a new file on automation controller called */etc/pki/ca-trust/source/anchors/automationhub-root.crt*. After the new file is created, run the command **update-ca-trust** to update the CA trust store with the new certificate.

On the Private automation hub

```
$ sudo -i
# cat /etc/pulp/certs/root.crt
(copy the contents of the file, including the lines with 'BEGIN CERTIFICATE' and
'END CERTIFICATE')
```

On automation controller

```
$ sudo -i
# vi /etc/pki/ca-trust/source/anchors/automationhub-root.crt
(paste the contents of the root.crt file from the {PrivateHubName} into the new file and write to disk)
# update-ca-trust
```

4.8. IMPORTING COLLECTIONS INTO PRIVATE AUTOMATION HUB

You can download collection tarball files from the following sources:

- Red Hat certified collections are found on Red Hat Automation Hub .
- Community collections are found on Ansible Galaxy.

4.8.1. Downloading collection from Red Hat Automation Hub

This section gives instructions on how to download a collection from Red Hat Automation Hub. If the collection has dependencies, they will also need to be downloaded and installed.

Procedure

1. Navigate to https://console.redhat.com/ansible/automation-hub/ and login with your Red Hat credentials.
2. Click on the **collection** you wish to download.
3. Click **Download tarball**
4. To verify if a collection has dependencies, click the **Dependencies** tab.
5. Download any dependencies needed for this collection.

4.9. CREATING COLLECTION NAMESPACE

The namespace of the collection must exist for the import to be successful. You can find the namespace name by looking at the first part of the collection tarball filename. For example the namespace of the collection *ansible-netcommon-3.0.0.tar.gz* is *ansible*.

Procedure

1. Login to private automation hub web console.
2. Navigate to **Collections → Namespaces**.
3. Click **Create**.
4. Provide the namespace name.

5. Click **Create**.

4.9.1. Importing the collection tarball with GUI

1. Login to private automation hub web console.
2. Navigate to **Collections → Namespaces**.
3. Click on **View collections** of the namespace you will be importing the collection into.
4. Click **Upload collection**.
5. Click the **folder icon** and select the tarball of the collection.
6. Click **Upload**.

This opens the 'My Imports' page. You can see the status of the import and various details of the files and modules that have been imported.

4.9.1.1. Importing the collection tarball using ansible-galaxy via CLI

You can import collections into the private automation hub by using the command-line interface rather than the GUI.

1. Copy the collection tarballs to the private automation hub.
2. Log in to the private automation hub server through SSH.
3. Add the self-signed root CA cert to the trust store on the automation hub.

```
# cp /etc/pulp/certs/root.crt \
    /etc/pki/ca-trust/source/anchors/automationhub-root.crt
# update-ca-trust
```

4. Update the **/etc/ansible/ansible.cfg** file with your hub configuration. Use either a token or a username and password for authentication.

```
[galaxy]
server_list = private_hub

[galaxy_server.private_hub]
url=https://<hub_fqdn>/api/galaxy/
token=<token_from_private_hub>
```

5. Import the collection using the ansible-galaxy command.

```
$ ansible-galaxy collection publish <collection_tarball>
```

NOTE

Create the namespace that the collection belongs to in advance or publishing the collection will fail.

4.10. APPROVING THE IMPORTED COLLECTION

After you have imported collections with either the GUI or the CLI method, you must approve them by using the GUI. After they are approved, they are available for use.

Procedure

1. Log in to private automation hub web console.
2. Go to **Collections → Approval**.
3. Click **Approve** for the collection you wish to approve.
4. The collection is now available for use in your private automation hub.

NOTE

The collection is added to the "Published" repository regardless of its source.

1. Import any dependency for the collection using these same steps.

Recommended collections depend on your use case. Ansible and Red Hat provide these collections.

4.10.1. Custom Execution Environments

Use the ansible-builder program to create custom execution environment images. For disconnected environments, custom EE images can be built in the following ways:

- Build an EE image on an internet-facing system and import it to the disconnected environment
- Build an EE image entirely on the disconnected environment with some modifications to the normal process of using ansible-builder
- Create a minimal base container image that includes all of the necessary modifications for a disconnected environment, then build custom EE images from the base container image

4.10.1.1. Transferring a Custom EE Images Across a Disconnected Boundary

A custom execution environment image can be built on an internet-facing machine using the existing documentation. Once an execution environment has been created it is available in the local Podman image cache. You can then transfer the custom EE image across a disconnected boundary. To transfer the custom EE image across a disconnected boundary, first save the image:

1. Save the image:

```
$ podman image save localhost/custom-ee:latest | gzip -c custom-ee-latest.tar.gz
```

Transfer the file across the disconnected boundary by using an existing mechanism such as sneakernet, one-way diode, etc.. After the image is available on the disconnected side, import it into the local podman cache, tag it, and push it to the disconnected hub:

```
$ podman image load -i custom-ee-latest.tar.gz
$ podman image tag localhost/custom-ee <hub_fqdn>/custom-ee:latest
$ podman login <hub_fqdn> --tls-verify=false
```

```
$ podman push <hub_fqdn>/custom-ee:latest
```

4.11. BUILDING AN EXECUTION ENVIRONMENT IN A DISCONNECTED ENVIRONMENT

When building a custom execution environment, the ansible-builder tool defaults to downloading the following requirements from the internet:

- Ansible Galaxy (galaxy.ansible.com) or Automation Hub (cloud.redhat.com) for any collections added to the EE image.
- PyPI (pypi.org) for any python packages required as collection dependencies.
- The UBI repositories (cdn.redhat.com) for updating any UBI-based EE images.
  - The RHEL repositories might also be needed to meet certain collection requirements.
- registry.redhat.io for access to the ansible-builder-rhel8 container image.

Building an EE image in a disconnected environment requires a subset of all of these mirrored, or otherwise made available on the disconnected network. See Importing Collections into Private Automation Hub for information about importing collections from Galaxy or Automation Hub into a private automation hub.

Mirrored PyPI content once transferred into the high-side network can be made available using a web server or an artifact repository like Nexus.

The UBI repositories can be mirrored on the low-side using a tool like **reposync**, imported to the disconnected environment, and made available from Satellite or a simple web server (since the content is freely redistributable).

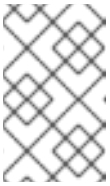
The **ansible-builder-rhel8** container image can be imported into a private automation hub in the same way a custom EE can be imported. See Transferring a Custom EE Images Across a Disconnected Boundary for details substituting **localhost/custom-ee** for **registry.redhat.io/ansible-automation-platform-21/ansible-builder-rhel8**. This will make the ansible-builder-rhel8 image available in the private automation hub registry along with the default EE images.

Once all of the prerequisites are available on the high-side network, ansible-builder and Podman can be used to create a custom execution environment image.

4.12. INSTALLING THE ANSIBLE-BUILDER RPM

Procedure

1. On a RHEL system, install the ansible-builder RPM. This can be done in one of several ways:
   a. Subscribe the RHEL box to a Satellite on the disconnected network.
   b. Attach the Ansible Automation Platform subscription and enable the Ansible Automation Platform repository.
   c. Install the ansible-builder RPM.



NOTE

This is preferred if a Satellite exists because the execution environment images can use RHEL content from the Satellite if the underlying build host is registered.

2. Unarchive the Ansible Automation Platform setup bundle.

3. Install the ansible-builder RPM and its dependencies from the included content:

```
$ tar -xzvf ansible-automation-platform-setup-bundle-2.3-1.2.tar.gz
$ cd ansible-automation-platform-setup-bundle-2.3-1.2/bundle/el8/repos/
$ sudo yum install ansible-builder-1.2.0-1.el9ap.noarch.rpm
python38-requirements-parser-0.2.0-4.el9ap.noarch.rpm
```

4. Create a directory for your custom EE build artifacts.

```
$ mkdir custom-ee
$ cd custom-ee/
```

5. Create an *execution-environment.yml* file that defines the requirements for your custom EE following the documentation at https://ansible-builder.readthedocs.io/en/stable/definition/. Override the **EE_BASE_IMAGE** and **EE_BUILDER_IMAGE** variables to point to the EEs available in your private automation hub.

```
$ cat execution-environment.yml
---
version: 1
build_arg_defaults:
  EE_BASE_IMAGE: '<hub_fqdn>/ee-supported-rhel8:latest'
  EE_BUILDER_IMAGE: '<hub_fqdn>/ansible-builder-rhel8:latest'

dependencies:
  python: requirements.txt
  galaxy: requirements.yml
```

6. Create an *ansible.cfg* file that points to your private automation hub and contains credentials that allow uploading, such as an admin user token.

```
$ cat ansible.cfg
[galaxy]
server_list = private_hub

[galaxy_server.private_hub]
url=https://<hub_fqdn>/api/galaxy/
token=<admin_token>
```

7. Create a *ubi.repo* file that points to your disconnected UBI repo mirror (this could be your Satellite if the UBI content is hosted there).
This is an example output where **reposync** was used to mirror the UBI repos.

```
$ cat ubi.repo
[ubi-8-baseos]
name = Red Hat Universal Base Image 8 (RPMs) - BaseOS
```

```
baseurl = http://<ubi_mirror_fqdn>/repos/ubi-8-baseos
enabled = 1
gpgkey = file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
gpgcheck = 1

[ubi-8-appstream]
name = Red Hat Universal Base Image 8 (RPMs) - AppStream
baseurl = http://<ubi_mirror_fqdn>/repos/ubi-8-appstream
enabled = 1
gpgkey = file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
gpgcheck = 1
```

8. Add the CA certificate used to sign the private automation hub web server certificate.

    a. For self-signed certificates (the installer default), make a copy of the file */etc/pulp/certs/root.crt* from your private automation hub and name it **hub-root.crt**.

    b. If an internal certificate authority was used to request and sign the private automation hub web server certificate, make a copy of that CA certificate called **hub-root.crt**.

9. Create your python requirements.txt and ansible collection requirements.yml with the content needed for your custom EE image. Note that any collections you require should already be uploaded into your private automation hub.

10. Use ansible-builder to create the context directory used to build the EE image.

```
$ ansible-builder create
Complete! The build context can be found at: /home/cloud-user/custom-ee/context
$ ls -1F
ansible.cfg
context/
execution-environment.yml
hub-root.crt
pip.conf
requirements.txt
requirements.yml
ubi.repo
```

11. Copy the files used to override the internet-facing defaults into the context directory.

```
$ cp ansible.cfg hub-root.crt pip.conf ubi.repo context/
```

12. Edit the file **context/Containerfile** and add the following modifications.

    a. In the first **EE_BASE_IMAGE** build section, add the **ansible.cfg** and **hub-root.crt** files and run the **update-ca-trust** command.

    b. In the **EE_BUILDER_IMAGE** build section, add the **ubi.repo** and **pip.conf** files.

    c. In the final **EE_BASE_IMAGE** build section, add the **ubi.repo** and pip.conf files.

```
$ cat context/Containerfile
ARG EE_BASE_IMAGE=<hub_fqdn>/ee-supported-rhel8:latest
ARG EE_BUILDER_IMAGE=<hub_fqdn>/ansible-builder-rhel8:latest

FROM $EE_BASE_IMAGE as galaxy
```

```
ARG ANSIBLE_GALAXY_CLI_COLLECTION_OPTS=
USER root

ADD _build /build
WORKDIR /build

# this section added
ADD ansible.cfg /etc/ansible/ansible.cfg
ADD hub-root.crt /etc/pki/ca-trust/source/anchors/hub-root.crt
RUN update-ca-trust
# end additions
RUN ansible-galaxy role install -r requirements.yml \
    --roles-path /usr/share/ansible/roles
RUN ansible-galaxy collection install \
    $ANSIBLE_GALAXY_CLI_COLLECTION_OPTS -r requirements.yml \
    --collections-path /usr/share/ansible/collections

FROM $EE_BUILDER_IMAGE as builder

COPY --from=galaxy /usr/share/ansible /usr/share/ansible

ADD _build/requirements.txt requirements.txt
RUN ansible-builder introspect --sanitize \
    --user-pip=requirements.txt \
    --write-bindep=/tmp/src/bindep.txt \
    --write-pip=/tmp/src/requirements.txt
# this section added
ADD ubi.repo /etc/yum.repos.d/ubi.repo
ADD pip.conf /etc/pip.conf
# end additions
RUN assemble

FROM $EE_BASE_IMAGE
USER root

COPY --from=galaxy /usr/share/ansible /usr/share/ansible
# this section added
ADD ubi.repo /etc/yum.repos.d/ubi.repo
ADD pip.conf /etc/pip.conf
# end additions

COPY --from=builder /output/ /output/
RUN /output/install-from-bindep && rm -rf /output/wheels
```

13. Create the EE image in the local podman cache using the **podman** command.

```
$ podman build -f context/Containerfile \
    -t <hub_fqdn>/custom-ee:latest
```

14. Once the custom EE image builds successfully, push it to the private automation hub.

```
$ podman push <hub_fqdn>/custom-ee:latest
```

4.12.1. Workflow for upgrading between minor Ansible Automation Platform releases

To upgrade between minor releases of Ansible Automation Platform 2, use this general workflow.

Procedure

1. Download and unarchive the latest Ansible Automation Platform 2 setup bundle.
2. Take a backup of the existing installation.
3. Copy the existing installation inventory file into the new setup bundle directory.
4. Run **./setup.sh** to upgrade the installation.

For example, to upgrade from version 2.2.0-7 to 2.3-1.2, make sure that both setup bundles are on the initial controller node where the installation occurred:

```
$ ls -1F
ansible-automation-platform-setup-bundle-2.2.0-7/
ansible-automation-platform-setup-bundle-2.2.0-7.tar.gz
ansible-automation-platform-setup-bundle-2.3-1.2/
ansible-automation-platform-setup-bundle-2.3-1.2.tar.gz
```

Back up the 2.2.0-7 installation:

```
$ cd ansible-automation-platform-setup-bundle-2.2.0-7
$ sudo ./setup.sh -b
$ cd ..
```

Copy the 2.2.0-7 inventory file into the 2.3-1.2 bundle directory:

```
$ cd ansible-automation-platform-setup-bundle-2.2.0-7
$ cp inventory ../ansible-automation-platform-setup-bundle-2.3-1.2/
$ cd ..
```

Upgrade from 2.2.0-7 to 2.3-1.2 with the setup.sh script:

```
$ cd ansible-automation-platform-setup-bundle-2.3-1.2
$ sudo ./setup.sh
```

APPENDIX A. INVENTORY FILE VARIABLES

The following tables contain information about the pre-defined variables used in Ansible installation inventory files. Not all of these variables are required.

A.1. GENERAL VARIABLES

Variable	Description
enable_insights_collection	The default install registers the node to the Red Hat Insights for Red Hat Ansible Automation Platform Service if the node is registered with Subscription Manager. Set to **False** to disable. Default = **true**
registry_password	**registry_password** is only required if a non-bundle installer is used. Password credential for access to **registry_url**. Used for both **[automationcontroller]** and **[automationhub]** groups. Enter your Red Hat Registry Service Account credentials in **registry_username** and **registry_password** to link to the Red Hat container registry. When **registry_url** is **registry.redhat.io**, username and password are required if not using bundle installer.
registry_url	Used for both **[automationcontroller]** and **[automationhub]** groups. Default = **registry.redhat.io**.
registry_username	**registry_username** is only required if a non-bundle installer is used. User credential for access to **registry_url**. Used for both **[automationcontroller]** and **[automationhub]** groups, but only if the value of **registry_url** is **registry.redhat.io**. Enter your Red Hat Registry Service Account credentials in **registry_username** and **registry_password** to link to the Red Hat container registry.

Variable	Description
routable_hostname	**routable hostname** is used if the machine running the installer can only route to the target host through a specific URL, for example, if you use shortnames in your inventory, but the node running the installer can only resolve that host using FQDN. If **routable_hostname** is not set, it should default to **ansible_host**. Then if, and only if **ansible_host** is not set, **inventory_hostname** is used as a last resort. Note that this variable is used as a host variable for particular hosts and not under the **[all:vars]** section. For further information, see Assigning a variable to one machine:host variables

A.2. ANSIBLE AUTOMATION HUB VARIABLES

Variable	Description
automationhub_admin_password	Required
automationhub_api_token	If upgrading from Ansible Automation Platform 2.0 or earlier, you must either: • provide an existing Ansible automation hub token as **automationhub_api_token**, or • set **generate_automationhub_token** to **true** to generate a new token Generating a new token invalidates the existing token.
automationhub_authentication_backend	This variable is not set by default. Set it to **ldap** to use LDAP authentication. When this is set to **ldap**, you must also set the following variables: • **automationhub_ldap_server_uri** • **automationhub_ldap_bind_dn** • **automationhub_ldap_bind_password** • **automationhub_ldap_user_search_base_dn** • **automationhub_ldap_group_search_base_dn**

Variable	Description
automationhub_auto_sign_collections	If a collection signing service is enabled, collections are not signed automatically by default. Setting this parameter to **true** signs them by default. Default = **false**.
automationhub_backup_collections	*Optional* Ansible automation hub provides artifacts in **/var/lib/pulp**. Automation controller automatically backs up the artifacts by default. You can also set **automationhub_backup_collections = false** and the backup/restore process does not then backup or restore **/var/lib/pulp**. Default = **true**
automationhub_collection_seed_repository	When the bundle installer is run, validated content is uploaded to the **validated** repository, and certified content is uploaded to the **rh-certified** repository. By default, both certified and validated content are uploaded. Possible values of this variable are 'certified' or 'validated'. If you do not want to install content, set **automationhub_seed_collections** to **false** to disable the seeding. If you only want one type of content, set **automationhub_seed_collections** to **true** and **automationhub_collection_seed_repository** to the type of content you do want to include.
automationhub_collection_signing_service_key	If a collection signing service is enabled, you must provide this variable to ensure that collections can be properly signed. **/absolute/path/to/key/to/sign**
automationhub_collection_signing_service_script	If a collection signing service is enabled, you must provide this variable to ensure that collections can be properly signed. **/absolute/path/to/script/that/signs**

Variable	Description
automationhub_create_default_collection_signing_service	The default install does not create a signing service. If set to **true** a signing service is created. Default = **false**
automationhub_container_signing_service_key	If a container signing service is enabled, you must provide this variable to ensure that containers can be properly signed. **/absolute/path/to/key/to/sign**
automationhub_container_signing_service_script	If a collection signing service is enabled, you must provide this variable to ensure that containers can be properly signed. **/absolute/path/to/script/that/signs**
automationhub_create_default_contaier_signing_service	The default install does not create a signing service. If set to **true** a signing service is created. Default = **false**
automationhub_disable_hsts	The default install deploys a TLS enabled Ansible automation hub. Use if automation hub is deployed with *HTTP Strict Transport Security* (HSTS) web-security policy enabled. Unless specified otherwise, the HSTS web-security policy mechanism is enabled. This setting allows you to disable it if required. Default = **false**
automationhub_disable_https	*Optional* If Ansible automation hub is deployed with HTTPS enabled. Default = **false**.
automationhub_enable_api_access_log	When set to **true**, creates a log file at **/var/log/galaxy_api_access.log** that logs all user actions made to the platform, including their username and IP address. Default = **false**.

Variable	Description
automationhub_enable_analytics	A Boolean indicating whether to enable pulp analytics for the version of pulpcore used in automation hub in Ansible Automation Platform 2.3. To enable pulp analytics, set **automationhub_enable_analytics = true**. Default = **false**.
automationhub_enable_unauthenticated_collection_access	Enables unauthorized users to view collections. To enable unauthorized users to view collections, set **automationhub_enable_unauthenticated_collection_access = true**. Default = **false**.
automation_hub_enable_unauthenticated_collection_download	Enables unauthorized users to download collections. To enable unauthorized users to download collections, set **automationhub_enable_unauthenticated_collection_download = true**. Default = **false**.
automationhub_importer_settings	*Optional* Dictionary of setting to pass to galaxy-importer. At import time collections can go through a series of checks. Behavior is driven by **galaxy-importer.cfg** configuration. Examples are **ansible-doc**, **ansible-lint**, and **flake8**. This parameter enables you to drive this configuration.

Variable	Description
automationhub_main_url	The main {HubNameShort} URL that clients connect to. For example, https://<load balancer host>. If not specified, the first node in the **[automationhub]** group is used. Use **automationhub_main_url** to specify the main automation hub URL that clients connect to if you are implementing Red Hat Single Sign-On on your automation hub environment.
automationhub_pg_database	*Required* The database name. Default = **automationhub**
automationhub_pg_host	Required if not using internal database.
automationhub_pg_password	The password for the automation hub PostgreSQL database. Do not use special characters for **automationhub_pg_password**. They can cause the password to fail.
automationhub_pg_port	Required if not using internal database. Default = 5432
automationhub_pg_sslmode	Required. Default = **prefer**
automationhub_pg_username	Required Default = **automationhub**

Variable	Description
automationhub_require_content_approval	*Optional* If automation hub enforces the approval mechanism before collections are made available. By default when you upload collections to automation hub an administrator must approve it before it is made available to the users. If you want to disable the content approval flow, set the variable to **false**. Default = **true**
automationhub_seed_collections	A boolean that defines whether or not preloading is enabled. When the bundle installer is run, by a new repository is created by default in private automation hub named **validated`** and the list of the validated collections is updated. If you do not want to install content, set **automationhub_seed_collections** to **false** to disable the seeding. If you only want one type of content, set **automationhub_seed_collections** to **true** and **automationhub_collection_seed_repository** to the type of content you do want to include. Default = **true**
automationhub_ssl_cert	*Optional* **/path/to/automationhub.cert** Same as **web_server_ssl_cert** but for automation hub UI and API
automationhub_ssl_key	*Optional* **/path/to/automationhub.key** Same as **web_server_ssl_key** but for automation hub UI and API

Variable	Description
automationhub_ssl_validate_certs	For Red Hat Ansible Automation Platform 2.3 and later, this value is no longer used. If automation hub should validate certificate when requesting itself because by default, Ansible Automation Platform deploys with self-signed certificates. Default = **false**.
automationhub_upgrade	**Deprecated** For Ansible Automation Platform 2.2.1 and later, the value of this has been fixed at true. Automation hub always updates with the latest packages.
generate_automationhub_token	If upgrading from Red Hat Ansible Automation Platform 2.0 or earlier, you must either: • provide an existing Ansible automation hub token as **automationhub_api_token** or • set **generate_automationhub_token** to **true** to generate a new token. Generating a new token will invalidate the existing token.
nginx_hsts_max_age	This variable specifies how long, in seconds, the system should be considered as a *HTTP Strict Transport Security* (HSTS) host. That is, how long HTTPS is used exclusively for communication. Default = 63072000 seconds, or two years.
nginx_tls_protocols	Defines support for **ssl_protocols** in Nginx. Default = **TLSv1.2**.
pulp_db_fields_key	Relative or absolute path to the Fernet symmetric encryption key you want to import. The path is on the Ansible management node. It is used to encrypt certain fields in the database (such as credentials.) If not specified, a new key will be generated.

For Ansible automation hub to connect to LDAP directly; the following variables must be configured. A list of other LDAP related variables (not covered by the **automationhub_ldap_xxx** variables below) that can be passed using the **ldap_extra_settings** variable can be found here: https://django-auth-ldap.readthedocs.io/en/latest/reference.html#settings

Variable	Description
automationhub_ldap_bind_dn	The name to use when binding to the LDAP server with **automationhub_ldap_bind_password**.
automationhub_ldap_bind_password	*Required* The password to use with **automationhub_ldap_bind_dn**.
automationhub_ldap_group_search_base_dn	An LDAPSearch object that finds all LDAP groups that users might belong to. If your configuration makes any references to LDAP groups, this and **automationhub_ldap_group_type** must be set. Default = **None**
automatiohub_ldap_group_search_filter	*Optional* Search filter for finding group membership. Variable identifies what objectClass type to use for mapping groups with automation hub and LDAP. Used for installing automation hub with LDAP. Default = **(objectClass=Group)**
automationhub_ldap_group_search_scope	*Optional* Scope to search for groups in an LDAP tree using the django framework for LDAP authentication. Used for installing automation hub with LDAP. Default = **SUBTREE**
automationhub_ldap_group_type_class	*Optional* Variable identifies the group type used during group searches within the django framework for LDAP authentication. Used for installing automation hub with LDAP. Default =**django_auth_ldap.config:GroupOfNamesType**
automationhub_ldap_server_uri	The URI of the LDAP server. This can be any URI that is supported by your underlying LDAP libraries.

Variable	Description
automationhub_ldap_user_search_base_dn	An LDAPSearch object that locates a user in the directory. The filter parameter should contain the placeholder %(user)s for the username. It must return exactly one result for authentication to succeed.
automationhub_ldap_user-search_scope	*Optional* Scope to search for users in an LDAP tree using django framework for LDAP authentication. Used for installing automation hub with LDAP. Default = `SUBTREE

A.3. RED HAT SINGLE SIGN-ON VARIABLES

*Use these variables for **automationhub** or **automationcatalog**.

Variable	Description
sso_automation_platform_login_theme	*Optional* Used for Ansible Automation Platform managed and externally managed Red Hat Single Sign-On. Path to the directory where theme files are located. If changing this variable, you must provide your own theme files. Default = **ansible-automation-platform**
sso_automation_platform_realm	*Optional* Used for Ansible Automation Platform managed and externally managed Red Hat Single Sign-On. The name of the realm in SSO. Default = **ansible-automation-platform**
sso_automation_platform_realm_displayname	*Optional* Used for Ansible Automation Platform managed and externally managed Red Hat Single Sign-On. Display name for the realm. Default = **Ansible Automation Platform**

Variable	Description
sso_console_admin_username	*Optional* Used for Ansible Automation Platform managed and externally managed Red Hat Single Sign-On. SSO administration username. Default = **admin**
sso_console_admin_password	*Required* Used for Ansible Automation Platform managed and externally managed Red Hat Single Sign-On. SSO administration password.
sso_custom_keystore_file	*Optional* Used for Ansible Automation Platform managed Red Hat Single Sign-On only. Customer-provided keystore for SSO.
sso_host	*Required* Used for Ansible Automation Platform externally managed Red Hat Single Sign-On only. Automation hub and Automation services catalog require SSO and SSO administration credentials for authentication. SSO administration credentials are also required to set automation services catalog specific roles needed for the application. If SSO is not provided in the inventory for configuration, then you must use this variable to define the SSO host.
sso_keystore_file_remote	*Optional* Used for Ansible Automation Platform managed Red Hat Single Sign-On only. Set to **true** if the customer-provided keystore is on a remote node. Default = **false**

Variable	Description
sso_keystore_name	*Optional* Used for Ansible Automation Platform managed Red Hat Single Sign-On only. Name of keystore for SSO. Default = **ansible-automation-platform**
sso_keystore_password	Password for keystore for HTTPS enabled SSO. Required when using Ansible Automation Platform managed SSO and when HTTPS is enabled. The default install deploys SSO with **sso_use_https=true**.
sso_redirect_host	*Optional* Used for Ansible Automation Platform managed and externally managed Red Hat Single Sign-On. If **sso_redirect_host** is set, it is used by the application to connect to SSO for authentication. This must be reachable from client machines.
sso_ssl_validate_certs	*Optional* Used for Ansible Automation Platform managed and externally managed Red Hat Single Sign-On. Set to **true** if the certificate is to be validated during connection. Default = **true**
sso_use_https	*Optional* Used for Ansible Automation Platform managed and externally managed Red Hat Single Sign-On. If Single Sign On uses https. Default = **true**

A.4. AUTOMATION SERVICES CATALOG VARIABLES

Variable	Description

Variable	Description
automationcatalog_controller_password	Used to generate a token from a controller host. Requires **automation_controller_main_url** to be defined as well.
automationcatalog_controller_token	Used for a pre-created OAuth token for automation controller. This token will be used instead of generating a token.
automationcatalog_controller_username	Used to generate a token from a controller host. Requires **automation_controller_main_url** to be defined as well.
automationcatalog_controller_verify_ssl	Used to enable or disable SSL validation from automation services catalog to automation controller. Default = **true**.
automationcatalog_disable_hsts	Used to enable or disable HSTS web-security policy for automation services catalog. Default = `false.
automationcatalog_disable_https	Used to enable or disable HSTS web-security policy for Services Catalog. Default = **false**.
automationcatalog_enable_analytics_collection	Used to control activation of analytics collection for automation services catalog
automationcatalog_main_url	Used by the Red Hat Single Sign-On host configuration if there is an alternative hostname that needs to be used between the SSO and automation services catalog host.
automationcatalog_pg_database	The postgres database URL for your automation services catalog.
automationcatalog_pg_host	The PostgreSQL host (database node) for your automation services catalog
automationcatalog_pg_password	The password for the PostgreSQL database of your automation services catalog. Do not use special characters for **automationcatalog_pg_password**. They can cause the password to fail.

Variable	Description
automationcatalog_pg_port	The PostgreSQL port to use for your automation services catalog. Default = 5432
automationcatalog_pg_username	The postgres ID for your automation services catalog.
automationcatalog_ssl_cert	Path to a custom provided SSL certificate file. Requires **automationcatalog_ssl_key** The internally managed CA signs and creates the certificate if not provided and https is left enabled.
automationcatalog_ssl_key	Path to a custom provided SSL certificate key file. Requires **automationcatalog_ssl_cert**. The internally managed CA signs and creates the certificate if not provided and https is left enabled.

A.5. AUTOMATION CONTROLLER VARIABLES

Variable	Description
admin_password	The password for an administration user to access the UI upon install completion.
automation_controller_main_url	For an alternative front end URL needed for SSO configuration with automation services catalog, provide the URL. Automation services catalog requires either Controller to be installed with automation controller, or a URL to an active and routable Controller server must be provided with this variable
automationcontroller_password	Password for your automation controller instance.
automationcontroller_username	Username for your automation controller instance.
nginx_http_port	The nginx HTTP server listens for inbound connections. Default = 80

Variable	Description
nginx_https_port	The nginx HTTPS server listens for secure connections. Default = 443
nginx_hsts_max_age	This variable specifies how long, in seconds, the system should be considered as a *HTTP Strict Transport Security* (HSTS) host. That is, how long HTTPS is used exclusively for communication. Default = 63072000 seconds, or two years.
nginx_tls_protocols	Defines support for **ssl_protocols** in Nginx. Default = **TLSv1.2**.
node_state	*Optional* The status of a node or group of nodes. Valid options are **active**, **deprovision** to remove a node from a cluster or **iso_migrate** to migrate a legacy isolated node to an execution node. Default = **active**.
node_type	For **[automationcontroller]** group. Two valid **node_types** can be assigned for this group. A **node_type=control** implies that the node only runs project and inventory updates, but not regular jobs. A **node_type=hybrid** has the ability to run everything. Default for this group = **hybrid**. For **[execution_nodes]** group Two valid node_types can be assigned for this group. A **node_type=hop** implies that the node forwards jobs to an execution node. A **node_type=execution** implies that the node can run jobs. Default for this group = **execution**.

Variable	Description
peers	*Optional* The **peers** variable is used to indicate which nodes a specific host or group connects to. Wherever the peers variable is defined, an outbound connection will be established to the specific host or group. This variable is used to add **tcp-peer** entries in the **receptor.conf** file used for establishing network connections with other nodes. See Peering The peers variable can be a comma-separated list of hosts and/or groups from the inventory. This is resolved into a set of hosts that is used to construct the **receptor.conf** file.
pg_database	The name of the postgres database. Default = **awx**.
pg_host	The postgreSQL host, which can be an externally managed database.
pg_password	The password for the postgreSQL database. Do not use special characters for **pg_password**. They can cause the password to fail. NOTE You no longer have to provide a **pg_hashed_password** in your inventory file at the time of installation because PostgreSQL 13 can now store user passwords more securely. When you supply **pg_password** in the inventory file for the installer, PostgreSQL uses the SCRAM-SHA-256 hash to secure that password as part of the installation process.
pg_port	The postgreSQL port to use. Default = 5432
pg_ssl_mode	One of **prefer** or **verify-full**. Set to **verify-full** for client-side enforced SSL. Default = **prefer**.
pg_username	Your postgres database username. Default = **awx**.

Variable	Description
postgres_ssl_cert	location of postgres ssl certificate. **/path/to/pgsql_ssl.cert**
postgres_ssl_key	location of postgres ssl key. **/path/to/pgsql_ssl.key**
postgres_use_cert	Location of postgres user certificate. **/path/to/pgsql.crt**
postgres_use_key	Location of postgres user key. **/path/to/pgsql.key**
postgres_use_ssl	If postgres is to use SSL.
receptor_listener_port	Port to use for recptor connection. Default = 27199.
supervisor_start_retry_count	When specified (no default value exists), adds **startretries = <value specified>** to the supervisor config file (/etc/supervisord.d/tower.ini). See program:x Section Values for further explanation about **startretries**.
web_server_ssl_cert	*Optional* **/path/to/webserver.cert** Same as **automationhub_ssl_cert** but for web server UI and API.
web_server_ssl_key	*Optional* **/path/to/webserver.key** Same as **automationhub_server_ssl_key** but for web server UI and API.

A.6. ANSIBLE VARIABLES

The following variables control how Ansible Automation Platform interacts with remote hosts.

Additional information on variables specific to certain plugins can be found at https://docs.ansible.com/ansible-core/devel/collections/ansible/builtin/index.html

A list of global configuration options can be found at https://docs.ansible.com/ansible-core/devel/reference_appendices/config.html

Variable	Description
ansible_connection	The connection plugin used for the task on the target host. This can be the name of any of ansible connection plugin. SSH protocol types are **smart**, **ssh** or **paramiko**. Default = **smart**
ansible_host	The ip or name of the target host to use instead of **inventory_hostname**.
ansible_port	The connection port number, if not, the default (22 for ssh).
ansible_user	The user name to use when connecting to the host.
ansible_password	The password to use to authenticate to the host. Never store this variable in plain text. Always use a vault.
ansible_ssh_private_key_file	Private key file used by ssh. Useful if using multiple keys and you do not want to use an SSH agent.
ansible_ssh_common_args	This setting is always appended to the default command line for **sftp**, **scp**, and **ssh**. Useful to configure a ProxyCommand for a certain host (or group).
ansible_sftp_extra_args	This setting is always appended to the default **sftp** command line.
ansible_scp_extra_args	This setting is always appended to the default **scp** command line.
ansible_ssh_extra_args	This setting is always appended to the default **ssh** command line.
ansible_ssh_pipelining	Determines if SSH pipelining is used. This can override the pipelining setting in **ansible.cfg**. If using SSH key-based authentication, then the key must be managed by an SSH agent.

Variable	Description
ansible_ssh_executable	(added in version 2.2) This setting overrides the default behavior to use the system ssh. This can override the ssh_executable setting in **ansible.cfg**.
ansible_shell_type	The shell type of the target system. You should not use this setting unless you have set the **ansible_shell_executable** to a non-Bourne (sh) compatible shell. By default commands are formatted using sh-style syntax. Setting this to **csh** or **fish** causes commands executed on target systems to follow the syntax of those shells instead.
ansible_shell_executable	This sets the shell that the ansible controller uses on the target machine, and overrides the executable in **ansible.cfg** which defaults to/**bin**/**sh**. You should only change if it is not possible to use /**bin**/**sh**, that is, if/**bin**/**sh** is not installed on the target machine or cannot be run from sudo.
inventory_hostname	This variable takes the hostname of the machine from the inventory script or the ansible configuration file. You cannot set the value of this variable. Because the value is taken from the configuration file, the actual runtime hostname value can vary from what is returned by this variable.