Red Hat Ansible Automation Platform 2.4

Red Hat Ansible Automation Platform installation guide

Install Ansible Automation Platform

Last Updated: 2024-09-27

Red Hat Ansible Automation Platform 2.4 Red Hat Ansible Automation Platform installation guide

Install Ansible Automation Platform

Legal Notice

Abstract

This guide shows you how to install Red Hat Ansible Automation Platform based on supported installation scenarios.

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.

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

If you have a suggestion to improve this documentation, or find an error, you can contact technical support at https://access.redhat.com to open a request.

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 by using a number of supported installation scenarios. Starting with Ansible Automation Platform 2.4, the installation scenarios include the optional deployment of Event-Driven Ansible controller, which introduces the automated resolution of IT requests.

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 by 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.

Verifying Event-Driven Ansible controller installation

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

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

To prevent errors, upgrade your RHEL nodes fully before installing Ansible Automation Platform.

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

Additional resources

For more information about obtaining a platform installer or system requirements, see 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 can obtain root access either through the **sudo** command, or through privilege escalation. For more on privilege escalation see Understanding privilege escalation .
- You can de-escalate privileges from root to users such as: AWX, PostgreSQL, Event-Driven Ansible, or Pulp.
- You have configured 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 8.8 or later 64-bit (x86, ppc64le, s390x, aarch64), or Red Hat Enterprise Linux 9.0 or later 64-bit (x86, ppc64le, s390x, aarch64)	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-core	Ansible-core version 2.14 or later	Ansible Automation Platform includes execution environments that contain ansible-core 2.15.
Python	3.9 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 network ports and protocols listed in *Table 5.3. Automation Hub* are available for successful connection and download of collections from automation hub or Ansible Galaxy server.
- Disable SSL inspection 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 Installing Ansible in the Ansible Community Documentation.

Additional notes for Red Hat Ansible Automation Platform requirements

- Red Hat Ansible Automation Platform depends on Ansible Playbooks and requires the installation of the latest stable version of ansible-core. You can download ansible-core manually or download it automatically as part of your installation of Red Hat Ansible Automation Platform.
- For new installations, automation controller installs the latest release package of ansible-core.
- If performing a bundled Ansible Automation Platform installation, the installation setup.sh script attempts to install ansible-core (and its dependencies) from the bundle for you.
- If you have installed Ansible manually, the Ansible Automation Platform installation setup.sh script detects that Ansible has been installed and does not attempt to reinstall it.

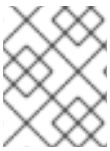
NOTE

You must install Ansible using a package manager such as **dnf**, 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 2.4 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, four node types are provided as abstractions to help you design the topology appropriate for your use case: control, hybrid, execution, and hop nodes.

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

Execution nodes run automation. Increase memory and CPU to increase capacity for running more forks.

NOTE

- The RAM and CPU resources stated might not be required for packages installed on an execution node but are the minimum recommended to handle the job load for a node to run an average number of jobs simultaneously.
- Recommended RAM and CPU node sizes are not supplied. The required RAM or CPU depends directly on the number of jobs you are running in that environment.

For further information about required RAM and CPU levels, see Performance tuning for automation controller.

Table 2.2. Execution nodes

Requirement	Minimum required
RAM	16 GB
CPUs	4
Local disk	40GB minimum

Control nodes

Control nodes process events and run cluster jobs including project updates and cleanup jobs. Increasing CPU and memory can help with job event processing.

Table 2.3. Control nodes

Requirement	Minimum 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.

Hop nodes

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

Table 2.4. Hop nodes

Requirement	Minimum required
RAM	16 GB
CPUs	4
Local disk	40 GB

- 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 and 2 GB reservation for automation controller. For more information, see Automation controller capacity determination and job impact. 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 Importing 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 Automation controller capacity determination and job impact.
CPUs	2 minimum	For capacity based on forks in your configuration, see Automation controller capacity determination and job impact.
Local disk	60 GB disk	Dedicate a minimum of 40GB to **/var** for collection storage.

IMPORTANT

Ansible automation execution nodes and automation hub system requirements are different and might not meet your network's needs. The general formula for determining how much memory you need is: Total control capacity = Total Memory in MB / Fork size in MB.

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 such as 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 about inventory file variables, see Inventory file variables in the *Red Hat Ansible Automation Platform Installation Guide*.

2.3.1. High availability automation hub requirements

Before deploying a high availability (HA) automation hub, ensure that you have a shared filesystem installed in your environment and that you have configured your network storage system, if applicable.

2.3.1.1. Required shared filesystem

A high availability automation hub requires you to have a shared file system, such as NFS, already installed in your environment. Before you run the Red Hat Ansible Automation Platform installer, verify that you installed the **/var/lib/pulp** directory across your cluster as part of the shared file system installation. The Red Hat Ansible Automation Platform installer returns an error if **/var/lib/pulp** is not detected in one of your nodes, causing your high availability automation hub setup to fail.

If you receive an error stating **/var/lib/pulp** is not detected in one of your nodes, ensure **/var/lib/pulp** is properly mounted in all servers and re-run the installer.

2.3.1.2. Installing firewalld for network storage

If you intend to install a HA automation hub using a network storage on the automation hub nodes itself, you must first install and use **firewalld** to open the necessary ports as required by your shared storage system before running the Ansible Automation Platform installer.

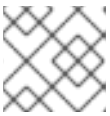
Install and configure **firewalld** by executing the following commands:

1. Install the **firewalld** daemon:

```
$ dnf install firewalld
```

2. Add your network storage under <service> using the following command:

```
$ firewall-cmd --permanent --add-service=<service>
```



NOTE

For a list of supported services, use the **$ firewall-cmd --get-services** command

3. Reload to apply the configuration:

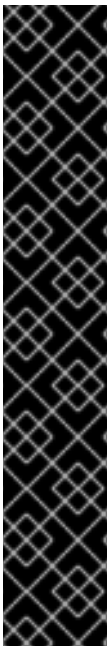
```
$ firewall-cmd --reload
```

2.4. EVENT-DRIVEN ANSIBLE CONTROLLER SYSTEM REQUIREMENTS

The Event-Driven Ansible controller is a single-node system capable of handling a variable number of long-running processes (such as rulebook activations) on-demand, depending on the number of CPU cores. Use the following minimum requirements to run, by default, a maximum of 12 simultaneous activations:

Requirement	Required
RAM	16 GB
CPUs	4

Requirement	Required
Local disk	40 GB minimum



IMPORTANT

- If you are running Red Hat Enterprise Linux 8 and want to set your memory limits, you must have cgroup v2 enabled before you install Event-Driven Ansible. For specific instructions, see the Knowledge-Centered Support (KCS) article, Ansible Automation Platform Event-Driven Ansible controller for Red Hat Enterprise Linux 8 requires cgroupv2.
- When you activate an Event-Driven Ansible rulebook under standard conditions, it uses about 250 MB of memory. However, the actual memory consumption can vary significantly based on the complexity of your rules and the volume and size of the events processed. In scenarios where a large number of events are anticipated or the rulebook complexity is high, conduct a preliminary assessment of resource usage in a staging environment. This ensures that your maximum number of activations is based on the capacity of your resources. See Single automation controller, single automation hub, and single Event-Driven Ansible controller node with external (installer managed) database for an example on setting Event-Driven Ansible controller maximum running activations.

2.5. 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** command.

Table 2.5. Database

Service	Required	Notes

Service	Required	Notes
Database	• 20 GB dedicated hard disk space • 4 CPUs • 16 GB RAM	• 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, the old job runs and facts must be cleaned on a regular basis. For more information, see Management Jobs in the *Automation Controller Administration Guide*.

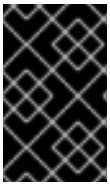
PostgreSQL Configurations

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. For more information about the settings you can use to improve database performance, see Database Settings.

Additional resources

For more information about tuning your PostgreSQL server, see the PostgreSQL documentation.

2.5.1. Setting up an external (customer supported) database



IMPORTANT

Red Hat does not support the use of external (customer supported) databases, however they are used by customers. The following guidance on inital configuration, from a product installation perspective only, is provided to avoid related support requests.

To create a database, user and password on an external PostgreSQL compliant database for use with automation controller, use the following procedure.

Procedure

1. Install and then connect to a PostgreSQL compliant database server with superuser privileges.

   ```
   # psql -h <db.example.com> -U superuser -p 5432 -d postgres <Password for user superuser>:
   ```

   Where:

   ```
   -h hostname
   --host=hostname
   ```

   Specifies the host name of the machine on which the server is running. If the value begins with a slash, it is used as the directory for the Unix-domain socket.

   ```
   -d dbname
   --dbname=dbname
   ```

   Specifies the name of the database to connect to. This is equivalent to specifying **dbname** as the first non-option argument on the command line. The **dbname** can be a connection string. If so, connection string parameters override any conflicting command line options.

   ```
   -U username
   --username=username
   ```

   Connect to the database as the user **username** instead of the default. (You must have permission to do so.)

2. Create the user, database, and password with the **createDB** or administrator role assigned to the user. For further information, see Database Roles.

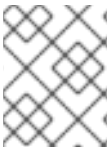
3. Add the database credentials and host details to the automation controller inventory file as an external database.
   The default values are used in the following example.

   ```
   [database]
   pg_host='db.example.com'
   pg_port=5432
   pg_database='awx'
   pg_username='awx'
   pg_password='redhat'
   ```

4. Run the installer.
   If you are using a PostgreSQL database with automation controller, the database is owned by the connecting user and must have a **createDB** or administrator role assigned to it.

5. Check that you are able to connect to the created database with the user, password and database name.

6. Check the permission of the user, the user should have the **createDB** or administrator role.



NOTE

During this procedure, you must check the External Database coverage. For further information, see https://access.redhat.com/articles/4010491

2.5.2. Enabling the hstore extension for the automation hub PostgreSQL database

From Ansible Automation Platform 2.4, the database migration script uses **hstore** fields to store information, therefore the **hstore** extension to the automation hub PostgreSQL database must be enabled.

This process is automatic when using the Ansible Automation Platform installer and a managed PostgreSQL server.

If the PostgreSQL database is external, you must enable the **hstore** extension to the automation hub PostreSQL database manually before automation hub installation.

If the **hstore** extension is not enabled before automation hub installation, a failure is raised during database migration.

Procedure

1. Check if the extension is available on the PostgreSQL server (automation hub database).

```
$ psql -d <automation hub database> -c "SELECT * FROM pg_available_extensions WHERE name='hstore'"
```

Where the default value for **<automation hub database>** is **automationhub**.

Example output with hstore available:

```
name  | default_version | installed_version |comment
------+-----------------+-------------------+----------------------------------------------------
 hstore | 1.7           |                   | data type for storing sets of (key, value) pairs
(1 row)
```

Example output with hstore not available:

```
 name | default_version | installed_version | comment
------+-----------------+-------------------+---------
(0 rows)
```

2. On a RHEL based server, the **hstore** extension is included in the **postgresql-contrib** RPM package, which is not installed automatically when installing the PostgreSQL server RPM package.
To install the RPM package, use the following command:

```
dnf install postgresql-contrib
```

3. Create the **hstore** PostgreSQL extension on the automation hub database with the following command:

```
$ psql -d <automation hub database> -c "CREATE EXTENSION hstore;"
```

The output of which is:

```
CREATE EXTENSION
```

4. In the following output, the **installed_version** field contains the **hstore** extension used, indicating that **hstore** is enabled.

```
name | default_version | installed_version | comment
-----+-----------------+-------------------+-----------------------------------------------------
hstore |    1.7     |      1.7         | data type for storing sets of (key, value) pairs
(1 row)
```

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

Check whether the minimum Ansible Automation Platform PostgreSQL database requirements are met by using the Flexible I/O Tester (FIO) tool. FIO is a tool used to benchmark read and write IOPS performance of the storage system.

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.
- You have adjusted the value of the **size** parameter. Adjusting this value reduces the amount of test data.

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. You can deploy automation controller with other automation platform components, such as automation hub and Event-Driven Ansible controller. 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 several 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. You can use one of the following as a basis for your own inventory file:

- Single automation controller with external (installer managed) database
- Single automation controller and single automation hub with external (installer managed) database
- Single automation controller, single automation hub, and single event-driven ansible controller node with external (installer managed) database

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. You can use one of the supported Installation scenario examples as the basis for your **inventory** file.

Additional resources

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

3.2. INVENTORY FILE EXAMPLES BASED ON INSTALLATION SCENARIOS

Red Hat supports several installation scenarios for Ansible Automation Platform. You can develop your own inventory files using the example files as a basis, or you can use the example closest to your preferred installation scenario.

3.2.1. Inventory file recommendations based on installation scenarios

Before selecting your installation method for Ansible Automation Platform, review the following recommendations. Familiarity with these recommendations will streamline the installation process.

- For Red Hat Ansible Automation Platform or automation hub: Add an automation hub host in the **[automationhub]** group.
- 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.
  The FQDN must not contain either the **-** or the _ symbols, as it will not be processed correctly.

  Do not use **localhost**.
- **admin** is the default user ID for the initial log in to Ansible Automation Platform and cannot be changed in the inventory file.
- Use of special characters for **pg_password** is limited. The **!**, **#**, **0** and **@** characters are supported. Use of other special characters 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.

3.2.1.1. 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.example.com

[database]
data.example.com

[all:vars]
admin_password='<password>'
pg_host='data.example.com'
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
```

3.2.1.2. Single automation controller and single automation hub with external (installer managed) database

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

```
[automationcontroller]
controller.example.com

[automationhub]
automationhub.example.com

[database]
data.example.com

[all:vars]
admin_password='<password>'
pg_host='data.example.com'
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>'

automationhub_admin_password= <PASSWORD>

automationhub_pg_host='data.example.com'
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

# 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
```

3.2.1.2.1. Connecting automation hub to a Red Hat Single Sign-On environment

You can configure the inventory file further to connect automation hub to a Red Hat Single Sign-On installation.

You must configure a different set of variables when connecting to a Red Hat Single Sign-On installation managed by Ansible Automation Platform than when connecting to an external Red Hat Single Sign-On installation.

For more information about these inventory variables, refer to the Installing and configuring central authentication for the Ansible Automation Platform.

3.2.1.3. High availability automation hub

Use the following examples to populate the inventory file to install a highly available automation hub. This inventory file includes a highly available automation hub with a clustered setup.

You can configure your HA deployment further to implement Red Hat Single Sign-On and enable a high availability deployment of automation hub on SELinux.

Specify database host IP

- Specify the IP address for your database host, using the **automation_pg_host** and **automation_pg_port** inventory variables. For example:

```
automationhub_pg_host='192.0.2.10'
automationhub_pg_port=5432
```

- Also specify the IP address for your database host in the [database] section, using the value in the **automationhub_pg_host** inventory variable:

```
[database]
192.0.2.10
```

List all instances in a clustered setup

- If installing a clustered setup, replace **localhost ansible_connection=local** in the [automationhub] section with the hostname or IP of all instances. For example:

```
[automationhub]
automationhub1.testing.ansible.com ansible_user=cloud-user ansible_host=192.0.2.18
automationhub2.testing.ansible.com ansible_user=cloud-user ansible_host=192.0.2.20
automationhub3.testing.ansible.com ansible_user=cloud-user ansible_host=192.0.2.22
```

Next steps

Check that the following directives are present in **/etc/pulp/settings.py** in each of the private automation hub servers:

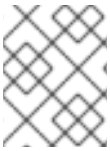
```
USE_X_FORWARDED_PORT = True
USE_X_FORWARDED_HOST = True
```

NOTE

If **automationhub_main_url** is not specified, the first node in the [automationhub] group will be used as default.

3.2.1.4. Enabling a high availability (HA) deployment of automation hub on SELinux

You can configure the inventory file to enable high availability deployment of automation hub on SELinux. You must create two mount points for **/var/lib/pulp** and **/var/lib/pulp/pulpcore_static**, and then assign the appropriate SELinux contexts to each.



NOTE

You must add the context for **/var/lib/pulp** pulpcore_static and run the Ansible Automation Platform installer before adding the context for **/var/lib/pulp**.

Prerequisites

- You have already configured a NFS export on your server.

Procedure

1. Create a mount point at **/var/lib/pulp**:

```
$ mkdir /var/lib/pulp/
```

2. Open **/etc/fstab** using a text editor, then add the following values:

```
srv_rhel8:/data /var/lib/pulp nfs
defaults,_netdev,nosharecache,context="system_u:object_r:var_lib_t:s0" 0 0
srv_rhel8:/data/pulpcore_static /var/lib/pulp/pulpcore_static nfs
defaults,_netdev,nosharecache,context="system_u:object_r:httpd_sys_content_rw_t:s0" 0 0
```

3. Run the reload systemd manager configuration command:

```
$ systemctl daemon-reload
```

4. Run the mount command for **/var/lib/pulp**:

```
$ mount /var/lib/pulp
```

5. Create a mount point at **/var/lib/pulp/pulpcore_static**:

```
$ mkdir /var/lib/pulp/pulpcore_static
```

6. Run the mount command:

```
$ mount -a
```

7. With the mount points set up, run the Ansible Automation Platform installer:

```
$ setup.sh -- -b --become-user root
```

8. After the installation is complete, unmount the **/var/lib/pulp/** mount point.

Next steps

1. Apply the appropriate SELinux context .
2. Configure the pulpcore.serivce.

Additional Resources

- See the SELinux Requirements on the Pulp Project documentation for a list of SELinux contexts.
- See the Filesystem Layout for a full description of Pulp folders.

3.2.1.4.1. Configuring pulpcore.service

After you have configured the inventory file, and applied the SELinux context, you now need to configure the pulp service.

Procedure

1. With the two mount points set up, shut down the Pulp service to configure **pulpcore.service**:

```
$ systemctl stop pulpcore.service
```

2. Edit **pulpcore.service** using **systemctl**:

```
$ systemctl edit pulpcore.service
```

3. Add the following entry to **pulpcore.service** to ensure that automation hub services starts only after starting the network and mounting the remote mount points:

```
[Unit]
After=network.target var-lib-pulp.mount
```

4. Enable **remote-fs.target**:

```
$ systemctl enable remote-fs.target
```

5. Reboot the system:

```
$ systemctl reboot
```

Troubleshooting

A bug in the pulpcore SELinux policies can cause the token authentication public/private keys in **etc/pulp/certs/** to not have the proper SELinux labels, causing the pulp process to fail. When this occurs, run the following command to temporarily attach the proper labels:

```
$ chcon system_u:object_r:pulpcore_etc_t:s0 /etc/pulp/certs/token_{private,public}_key.pem
```

Repeat this command to reattach the proper SELinux labels whenever you relabel your system.

3.2.1.4.2. Applying the SELinux context

After you have configured the inventory file, you must now apply the context to enable the high availability (HA) deployment of automation hub on SELinux.

Procedure

1. Shut down the Pulp service:

```
$ systemctl stop pulpcore.service
```

2. Unmount **/var/lib/pulp/pulpcore_static**:

```
$ umount /var/lib/pulp/pulpcore_static
```

3. Unmount **/var/lib/pulp/**:

```
$ umount /var/lib/pulp/
```

4. Open **/etc/fstab** using a text editor, then replace the existing value for **/var/lib/pulp** with the following:

```
srv_rhel8:/data /var/lib/pulp nfs
defaults,_netdev,nosharecache,context="system_u:object_r:pulpcore_var_lib_t:s0" 0 0
```

5. Run the mount command:

```
$ mount -a
```

3.2.1.5. Configuring content signing on private automation hub

To successfully sign and publish Ansible Certified Content Collections, you must configure private automation hub for signing.

Prerequisites

- Your GnuPG key pairs have been securely set up and managed by your organization.
- Your public-private key pair has proper access for configuring content signing on private automation hub.

Procedure

1. Create a signing script that accepts only a filename.

NOTE

This script acts as the signing service and must generate an ascii-armored detached **gpg** signature for that file using the key specified through the **PULP_SIGNING_KEY_FINGERPRINT** environment variable.

The script prints out a JSON structure with the following format.

```
{"file": "filename", "signature": "filename.asc"}
```

All the file names are relative paths inside the current working directory. The file name must remain the same for the detached signature.

Example:

The following script produces signatures for content:

```
#!/usr/bin/env bash

FILE_PATH=$1
SIGNATURE_PATH="$1.asc"

ADMIN_ID="$PULP_SIGNING_KEY_FINGERPRINT"
PASSWORD="password"

# Create a detached signature
gpg --quiet --batch --pinentry-mode loopback --yes --passphrase \
  $PASSWORD --homedir ~/.gnupg/ --detach-sign --default-key $ADMIN_ID \
  --armor --output $SIGNATURE_PATH $FILE_PATH

# Check the exit status
STATUS=$?
if [ $STATUS -eq 0 ]; then
```

```
    echo {\"file\": \"$FILE_PATH\", \"signature\": \"$SIGNATURE_PATH\"}
else
    exit $STATUS
fi
```

After you deploy a private automation hub with signing enabled to your Ansible Automation Platform cluster, new UI additions are displayed in collections.

2. Review the Ansible Automation Platform installer inventory file for options that begin with **automationhub_***.

```
[all:vars]
.
.
.
automationhub_create_default_collection_signing_service = True
automationhub_auto_sign_collections = True
automationhub_require_content_approval = True
automationhub_collection_signing_service_key = /abs/path/to/galaxy_signing_service.gpg
automationhub_collection_signing_service_script = /abs/path/to/collection_signing.sh
```

The two new keys (**automationhub_auto_sign_collections** and **automationhub_require_content_approval**) indicate that the collections must be signed and approved after they are uploaded to private automation hub.

3.2.1.6. 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 cannot complete the installation.

3.2.1.6.1. 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.

3.2.1.6.2. 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'
...
```

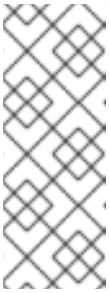
  - To deny private automation hub access (for example, members of the **cn=pah-nosoupforyou,ou=groups,dc=example,dc=com** group):

```
#ldapextras.yml
---
ldap_extra_settings:
  AUTH_LDAP_DENY_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.

3.2.1.6.3. LDAP referrals

If your LDAP servers return referrals, you might have to disable referrals to successfully authenticate using LDAP on private automation hub.

If not, the following message is returned:

```
Operation unavailable without authentication
```

To disable the LDAP REFERRALS lookup, set:

```
GALAXY_LDAP_DISABLE_REFERRALS = true
```

This sets **AUTH_LDAP_CONNECTIONS_OPTIONS** to the correct option.

3.2.1.7. Single automation controller, single automation hub, and single Event-Driven Ansible controller node with external (installer managed) database

Use this example to populate the inventory file to deploy single instances of automation controller, automation hub, and Event-Driven Ansible controller with an external (installer managed) database.

IMPORTANT

- This scenario requires a minimum of automation controller 2.4 for successful deployment of Event-Driven Ansible controller.
- Event-Driven Ansible controller must be installed on a separate server and cannot be installed on the same host as automation hub and automation controller.
- When you activate an Event-Driven Ansible rulebook under standard conditions, it uses approximately 250 MB of memory. However, the actual memory consumption can vary significantly based on the complexity of your rules and the volume and size of the events processed. In scenarios where a large number of events are anticipated or the rulebook complexity is high, conduct a preliminary assessment of resource usage in a staging environment. This ensures that your maximum number of activations is based on the capacity of your resources. In the following example, the default **automationedacontroller_max_running_activations** setting is 12, but you can adjust according to your capacity.

```
[automationcontroller]
controller.example.com

[automationhub]
automationhub.example.com

[automationedacontroller]
automationedacontroller.example.com

[database]
data.example.com

[all:vars]
admin_password='<password>'
pg_host='data.example.com'
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>'

# Automation hub configuration

automationhub_admin_password= <PASSWORD>

automationhub_pg_host='data.example.com'
automationhub_pg_port=5432

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

```
# Automation Event-Driven Ansible controller configuration

automationedacontroller_admin_password='<eda-password>'

automationedacontroller_pg_host='data.example.com'
automationedacontroller_pg_port=5432

automationedacontroller_pg_database='automationedacontroller'
automationedacontroller_pg_username='automationedacontroller'
automationedacontroller_pg_password='<password>'

# Keystore file to install in SSO node
# sso_custom_keystore_file='/path/to/sso.jks'

# This install will deploy SSO with sso_use_https=True
# Keystore password is required for https enabled SSO
sso_keystore_password=''

# This 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

# 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

# Boolean flag used to verify Automation Controller's
# web certificates when making calls from Automation Event-Driven Ansible controller.
# automationedacontroller_controller_verify_ssl = true
#
# Certificate and key to install in Automation Event-Driven Ansible controller node
# automationedacontroller_ssl_cert=/path/to/automationeda.crt
# automationedacontroller_ssl_key=/path/to/automationeda.key
```

3.2.1.8. Adding a safe plugin variable to Event-Driven Ansible controller

When using redhat.insights_eda or similar plug-ins to run rulebook activations in Event-Driven Ansible controller, you must add a safe plugin variable to a directory in Ansible Automation Platform. This would

ensure connection between Event-Driven Ansible controller and the source plugin, and display port mappings correctly.

Procedure

1. Create a directory for the safe plugin variable: **mkdir -p ./group_vars/automationedacontroller**
2. Create a file within that directory for your new setting (for example, **touch ./group_vars/automationedacontroller/custom.yml**)
3. Add the variable **automationedacontroller_safe_plugins** to the file with a comma-separated list of plugins to enable for Event-Driven Ansible controller. For example:

```
automationedacontroller_safe_plugins: “ansible.eda.webhook, ansible.eda.alertmanager”
```

3.3. 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.

3.4. VERIFYING INSTALLATION OF AUTOMATION CONTROLLER

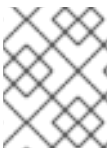
Verify that you installed automation controller successfully by logging in with the admin credentials you inserted in the **inventory** file.

Prerequisite

- Port 443 is available

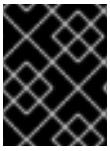
Procedure

1. Go to the IP address specified for the automation controller node in the **inventory** file.
2. Enter your Red Hat Satellite credentials. If this is your first time logging in after installation, upload your **manifest** file.
3. Log in with the user ID **admin** and the password 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.



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.

Upon a successful log in to automation controller, your installation of Red Hat Ansible Automation Platform 2.4 is complete.

3.4.1. Additional automation controller configuration and resources

See the following resources to explore additional automation controller configurations.

Table 3.1. Resources to configure automation controller

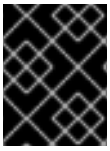
Resource link	Description
Automation Controller Quick Setup Guide	Set up automation controller and run your first playbook.
Automation Controller Administration Guide	Configure automation controller administration through customer scripts, management jobs, etc.
Configuring proxy support for Red Hat Ansible 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.

3.5. 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

1. Navigate to the IP address specified for the automation hub node in the **inventory** file.
2. Enter your Red Hat Satellite credentials. If this is your first time logging in after installation, upload your **manifest** file.
3. Log in with the user ID **admin** and the password 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.

Upon a successful login to automation hub, your installation of Red Hat Ansible Automation Platform 2.4 is complete.

3.5.1. Additional automation hub configuration and resources

See the following resources to explore additional automation hub configurations.

Table 3.2. Resources to configure automation controller

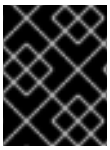
Resource link	Description
Managing user access in private automation hub	Configure user access for automation hub.
Managing Red Hat Certified, validated, and Ansible Galaxy content in automation hub	Add content to your automation hub.
Publishing proprietary content collections in automation hub	Publish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

3.6. VERIFYING EVENT-DRIVEN ANSIBLE CONTROLLER INSTALLATION

Verify that you installed Event-Driven Ansible 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 Event-Driven Ansible controller node in the **inventory** file.
2. Enter your Red Hat Satellite credentials. If this is your first time logging in after installation, upload your **manifest** file.
3. Log in with the user ID **admin** and the password 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 .

Upon a successful login to Event-Driven Ansible controller, your installation of Red Hat Ansible Automation Platform 2.4 is complete.

CHAPTER 4. DISCONNECTED INSTALLATION

If you are not connected to the internet or do not have access to online repositories, you can install Red Hat Ansible Automation Platform without an active internet connection.

4.1. PREREQUISITES

Before installing Ansible Automation Platform on a disconnected network, you must meet the following prerequisites:

1. A created subscription manifest. See Obtaining a manifest file for more information.
2. The Ansible Automation Platform setup bundle at Customer Portal is downloaded.
3. The DNS records for the automation controller and private automation hub servers are created.

4.2. ANSIBLE AUTOMATION PLATFORM INSTALLATION ON DISCONNECTED RHEL

You can install Ansible Automation Platform automation controller and private automation hub without an internet connection by using the installer-managed database located on the automation controller. Use the setup bundle for a disconnected installation as it includes additional components that make installing Ansible Automation Platform easier in a disconnected environment. These include the Ansible Automation Platform Red Hat package managers (RPMs) and the default execution environment (EE) images.

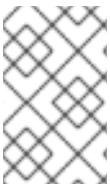
4.2.1. System requirements for disconnected installation

Ensure that your system has all the hardware requirements before performing a disconnected installation of Ansible Automation Platform. For more information about hardware requirements, see Chapter 2. System requirements.

4.2.2. 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 first obtain access to BaseOS and AppStream repositories. Use Satellite to sync repositories and add dependencies. If you prefer an alternative tool, you can choose between the following options:

- Reposync
- The RHEL Binary DVD



NOTE

The RHEL Binary DVD method requires the DVD for supported versions of RHEL, including version 8.6 or higher. See Red Hat Enterprise Linux Life Cycle for information about which versions of RHEL are currently supported.

Additional Resources

- Satellite

4.3. SYNCHRONIZING RPM REPOSITORIES 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. Use reposync with **--download-metadata** and without **--newest-only**. See RHEL 8 Reposync.

      - If you are not using **--newest-only,** the repos downloaded will be ~90GB.
      - If you are using **--newest-only,** the repos downloaded will be ~14GB.

3. If you plan to use Red Hat Single Sign-On, 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

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

4. Move the repositories to your disconnected network.

4.4. CREATING A NEW WEB SERVER TO HOST REPOSITORIES

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

Procedure

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*, and 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
```

4.5. ACCESSING RPM REPOSITORIES FROM A LOCALLY MOUNTED DVD

If you plan to access the repositories from the RHEL binary DVD, you must first set up a local repository.

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
```

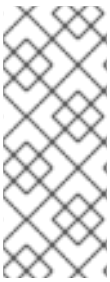
2. Create yum repo file at **/etc/yum.repos.d/dvd.repo**

```
[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]
```

Additional Resources

For further detail on setting up a repository see Need to set up yum repository for locally-mounted DVD on Red Hat Enterprise Linux 8.

4.6. 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. Log in to the Red Hat Customer Portal .
2. From the menu bar, select **Subscriptions** and select the **Subscriptions Allocations** tab .

3. Click **New subscription allocation**.
4. Name the new subscription allocation.
5. Select **Satellite 6.8** from the **Type** list.
6. Click **Create**. The **Details** tab opens for your subscription allocation.
7. Select the **Subscriptions** tab.
8. Click **Add Subscriptions**.
9. Find your Ansible Automation Platform subscription, and 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 will be managed by Ansible Automation Platform: server, network device, etc.
10. Click **Submit**.
11. Click **Export Manifest**.

This downloads a file *manifest_<allocation name>_<date>.zip* that be imported with automation controller after installation.

4.7. DOWNLOADING AND INSTALLING THE ANSIBLE AUTOMATION PLATFORM SETUP BUNDLE

Choose the setup bundle to download Ansible Automation Platform for disconnected installations. This bundle includes the RPM content for Ansible Automation Platform and the default execution environment images that will be uploaded to your private automation hub during the installation process.

Procedure

1. Download the Ansible Automation Platform setup bundle package by navigating to the Red Hat Ansible Automation Platform download page and clicking **Download Now** for the Ansible Automation Platform 2.4 Setup Bundle.
2. From automation controller, untar the bundle:

```
$ tar xvf \
  ansible-automation-platform-setup-bundle-2.4-1.tar.gz
$ cd ansible-automation-platform-setup-bundle-2.4-1
```

3. 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 file

```
[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'
```

4. Run the Ansible Automation Platform setup bundle executable as the root user:

```
$ sudo -i
# cd /path/to/ansible-automation-platform-setup-bundle-2.4-1
# ./setup.sh
```

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

6. Log in using the administrator credentials specified in the installation inventory file.

NOTE

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

4.8. COMPLETING POST INSTALLATION TASKS

After you have completed the installation of Ansible Automation Platform, ensure that automation hub and automation controller deploy properly.

4.8.1. Adding a controller subscription

Procedure

1. Navigate to the FQDN of the Automation controller. Log in with the username 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 must be turned off.
5. Click **Next**.
6. Read the End User License Agreement and click **Submit** if you agree.

4.8.2. Updating the CA trust store

As part of your post-installation tasks, you must update the software's certificates. By default, Ansible Automation Platform automation hub and automation controller are installed using self-signed certificates. Because of this, the controller does not trust the hub's certificate and will not download the execution environment from the hub.

To ensure that automation controller downloads the execution environment from automation hub, you must import the hub's Certificate Authority (CA) certificate as a trusted certificate on the controller. You can do this in one of two ways, depending on whether SSH is available as root user between automation controller and private automation hub.

4.8.2.1. Using secure copy (SCP) as a root user

If SSH is available as the root user between the controller and private automation hub, use SCP to copy the root certificate on the private automation hub to the controller.

Procedure

1. Run **update-ca-trust** on the controller to update the CA trust store:

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

4.8.2.2. Copying and pasting as a non root user

If SSH is unavailable as root between the private automation hub and the controller, copy the contents of the file */etc/pulp/certs/root.crt* on the private automation hub and paste it into a new file on the controller called */etc/pki/ca-trust/source/anchors/automationhub-root.crt*.

Procedure

1. Run **update-ca-trust** to update the CA trust store with the new certificate. On the private automation hub, run:

```
$ sudo -i
```

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

1. On the automation controller:

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

Additional Resources

- For further information on unknown certificate authority, see Project sync fails with unknown certificate authority error in Ansible Automation Platform 2.1.

4.9. IMPORTING COLLECTIONS INTO PRIVATE AUTOMATION HUB

You can download a collection as a tarball file from Ansible automation hub for use in your private automation hub. Certified collections are available on the automation hub Hybrid Cloud Console , and community collections are on Ansible Galaxy. You must also download and install any dependencies needed for the collection.

Procedure

1. Navigate to link: console.redhat.com and log in with your Red Hat credentials.
2. Click on the **collection** you want 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.

4.10. CREATING A COLLECTION NAMESPACE

Before importing a collection, you must first create a namespace for the collection in your private automation hub. 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. Log in to the automation hub Hybrid Cloud Console .
2. From the navigation panel, select **Collections → Namespaces**.
3. Click **Create**.
4. Provide the namespace name.
5. Click **Create**.

4.10.1. Importing the collection tarball by using the web console

Once the namespace has been created, you can import the collection by using the web console.

Procedure

1. Log in to automation hub Hybrid Cloud Console .
2. From the navigation panel, select **Collections → Namespaces**.
3. Click **View collections** next to 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.

4.10.2. Importing the collection tarball by using the CLI

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

Procedure

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

4.11. APPROVING THE IMPORTED COLLECTIONS

After you have imported collections by using 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 automation hub Hybrid Cloud Console .
2. From the navigation panel, select **Collections → Approval**.
3. Click **Approve** for the collection you want to approve.
4. The collection is now available for use in your private automation hub.
5. Import any dependency for the collection by repeating steps 2 and 3.

NOTE

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

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

4.11.1. Custom automation execution environments

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

- Build an execution environment image on an internet-facing system and import it to the disconnected environment.
- Build an execution environment 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 execution environment images from the base container image.

4.11.1.1. Transferring custom execution environment images across a disconnected boundary

You can build a custom execution environment image on an internet-facing machine. After you create an execution environment, it is available in the local podman image cache. You can then transfer the custom execution environment image across a disconnected boundary.

Procedure

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 or one-way diode.

2. 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
```

4.12. BUILDING AN EXECUTION ENVIRONMENT IN A DISCONNECTED ENVIRONMENT

Creating execution environments for Ansible Automation Platform is a common task which works differently in disconnected environments. When building a custom execution environment, the ansible-builder tool defaults to downloading content from the following locations on the internet:

- Red Hat Automation hub (console.redhat.com) or Ansible Galaxy (galaxy.ansible.com) for any Ansible content collections added to the execution environment image.
- PyPI (pypi.org) for any python packages required as collection dependencies.
- RPM repositories such as the RHEL or UBI repositories (cdn.redhat.com) for adding or updating RPMs to the execution environment image, if needed.
- registry.redhat.io for access to the base container images.

Building an execution environment image in a disconnected environment requires mirroring content from these locations. See Importing Collections into private automation hub for information about importing collections from Ansible Galaxy or automation hub into a private automation hub.

Mirrored PyPI content once transferred into the disconnected network can be made available by using a web server or an artifact repository such as Nexus. The RHEL and UBI repository content can be exported from an internet-facing Red Hat Satellite Server, copied into the disconnected environment, then imported into a disconnected Satellite so it is available for building custom execution environments. See ISS Export Sync in an Air-Gapped Scenario for details.

The default base container image, ee-minimal-rhel8, is used to create custom execution environment images and is included with the bundled installer. This image is added to the private automation hub at install time. If a different base container image such as ee-minimal-rhel9 is required, it must be imported to the disconnected network and added to the private automation hub container registry.

Once all of the prerequisites are available on the disconnected network, the ansible-builder command can be used to create custom execution environment images.

4.12.1. Installing the Ansible Builder RPM

On the RHEL system where custom execution environments will be built, you will install the Ansible Builder RPM by using a Satellite Server that already exists in the environment. This method is preferred because the execution environment images can use any RHEL content from the pre-existing Satellite if required.

Procedure

1. Install the Ansible Builder RPM from the Ansible Automation Platform repository.
   a. Subscribe the RHEL system to a Satellite on the disconnected network.
   b. Attach the Ansible Automation Platform subscription and enable the Ansible Automation Platform repository. The repository name is either **ansible-automation-platform-2.4-for-**

rhel-8-x86_64-rpms or **ansible-automation-platform-2.4-for-rhel-9-x86_64-rpms** depending on the version of RHEL used on the underlying system.

c. Install the Ansible Builder RPM. The version of the Ansible Builder RPM must be 3.0.0 or later in order for the examples below to work properly.

2. Install the Ansible Builder RPM from the Ansible Automation Platform setup bundle. Use this method if a Satellite Server is not available on your disconnected network.

a. Unarchive the Ansible Automation Platform setup bundle.

b. Install the Ansible Builder RPM and its dependencies from the included content.

```
$ tar -xzvf ansible-automation-platform-setup-bundle-2.4-3-x86_64.tar.gz
$ cd ansible-automation-platform-setup-bundle-2.4-3-x86_64/bundle/packages/el8/repos/
$ sudo dnf install ansible-builder-3.0.0-2.el8ap.noarch.rpm \
    python39-requirements-parser-0.2.0-4.el8ap.noarch.rpm \
    python39-bindep-2.10.2-3.el8ap.noarch.rpm \
    python39-jsonschema-4.16.0-1.el8ap.noarch.rpm \
    python39-pbr-5.8.1-2.el8ap.noarch.rpm \
    python39-distro-1.6.0-3.el8pc.noarch.rpm \
    python39-packaging-21.3-2.el8ap.noarch.rpm \
    python39-parsley-1.3-2.el8pc.noarch.rpm \
    python39-attrs-21.4.0-2.el8pc.noarch.rpm \
    python39-pyrsistent-0.18.1-2.el8ap.x86_64.rpm \
    python39-pyparsing-3.0.9-1.el8ap.noarch.rpm
```

NOTE

The specific versions may be slightly different depending on the version of the setup bundle being used.

Additional resources

- For details on creating a Satellite environment on a disconnected network see Installing Satellite Server in a Disconnected Network Environment.

4.12.2. Creating the custom execution environment definition

Once the Ansible Builder RPM is installed, use the following steps to create your custom execution environment.

1. Create a directory for the build artifacts used when creating your custom execution environment. Any new files created with the steps below will be created under this directory.

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

2. Create an **execution-environment.yml** file that defines the requirements for your custom execution environment.

NOTE

Version 3 of the execution environment definition format is required, so ensure the **execution-environment.yml** file contains **version: 3** explicitly before continuing.

a. Override the base image to point to the minimal execution environment available in your private automation hub.

b. Define the additional build files needed to point to any disconnected content sources that will be used in the build process. Your custom **execution-environment.yml** file should look similar to the following example:

```
$ cat execution-environment.yml
---
version: 3

images:
  base_image:
    name: private-hub.example.com/ee-minimal-rhel8:latest

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

additional_build_files:
  - src: files/ansible.cfg
    dest: configs
  - src: files/pip.conf
    dest: configs
  - src: files/hub-ca.crt
    dest: configs
  # uncomment if custom RPM repositories are required
  #- src: files/custom.repo
  #  dest: configs

additional_build_steps:
  prepend_base:
    # copy a custom pip.conf to override the location of the PyPI content
    - ADD _build/configs/pip.conf /etc/pip.conf
    # remove the default UBI repository definition
    - RUN rm -f /etc/yum.repos.d/ubi.repo
    # copy the hub CA certificate and update the trust store
    - ADD _build/configs/hub-ca.crt /etc/pki/ca-trust/source/anchors
    - RUN update-ca-trust
    # if needed, uncomment to add a custom RPM repository configuration
    #- ADD _build/configs/custom.repo /etc/yum.repos.d/custom.repo

  prepend_galaxy:
    - ADD _build/configs/ansible.cfg ~/.ansible.cfg

...
```

3. Create an **ansible.cfg** file under the **files/** subdirectory that points to your private automation hub.

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

[galaxy_server.private_hub]
url = https://private-hub.example.com/api/galaxy/
```

4. Create a **pip.conf** file under the **files/** subdirectory which points to the internal PyPI mirror (a web server or something like Nexus):

```
$ cat files/pip.conf
[global]
index-url = https://<pypi_mirror_fqdn>/
trusted-host = <pypi_mirror_fqdn>
```

5. Optional: If you use a **bindep.txt** file to add RPMs the custom execution environment, create a **custom.repo** file under the **files/** subdirectory that points to your disconnected Satellite or other location hosting the RPM repositories. If this step is necessary, uncomment the steps in the example **execution-environment.yml** file that correspond with the **custom.repo** file. The following example is for the UBI repos. Other local repos can be added to this file as well. The URL path may need to change depending on where the mirror content is located on the web server.

```
$ cat files/custom.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
```

6. Add the CA certificate used to sign the private automation hub web server certificate. If the private automation hub uses self-signed certificates provided by the installer:

   a. Copy the file **/etc/pulp/certs/pulp_webserver.crt** from your private automation hub and name it **hub-ca.crt**.

   b. Add the **hub-ca.crt** file to the **files/** subdirectory.

7. If the private automation hub uses user-provided certificates signed by a certificate authority:

   a. Make a copy of that CA certificate and name it **hub-ca.crt**.

   b. Add the **hub-ca.crt** file to the **files/** subdirectory.

8. Once the preceeding steps have been completed, create your python **requirements.txt** and Ansible collection **requirements.yml** files, with the content needed for your custom execution environment image.

NOTE

Any required collections must already be uploaded into your private automation hub.

The following files should exist under the **custom-ee/** directory, with **bindep.txt** and **files/custom.repo** being optional:

```
$ cd $HOME/custom-ee
$ tree .
.
├── bindep.txt
├── execution-environment.yml
├── files
│   ├── ansible.cfg
│   ├── custom.repo
│   ├── hub-ca.crt
│   └── pip.conf
├── requirements.txt
└── requirements.yml

1 directory, 8 files
```

Additional resources

For more information on the Version 3 format and requirements, see Execution Environment Definition: Version 3 Format.

4.12.3. Building the custom execution environment

Before creating the new custom execution environment, an API token from the private hub will be needed in order to download content.

Generate a token by taking the following steps:

1. Log in to your private hub.
2. Choose "Collections" from the left-hand menu.
3. Choose the"API token" under the "Collections" section of the menu.
4. Once you have the token, set the following environment variable so that Ansible Builder can access the token:

```
$ export ANSIBLE_GALAXY_SERVER_PRIVATE_HUB_TOKEN=<your_token>
```

5. Create the custom execution environment by using the command:

```
$ cd $HOME/custom-ee
$ ansible-builder build -f execution-environment.yml -t private-hub.example.com/custom-ee:latest -v 3
```

NOTE

If the build fails with an error that the private hub certificate is signed by an unknown authority, you can pull the required image into the local image cache by running the command:

```
$ podman pull private-hub.example.com/ee-minimal-rhel8:latest --tls-verify=false
```

Alternately, you can add the private hub CA certificate to the podman certificate store:

```
$ sudo mkdir /etc/containers/certs.d/private-hub.example.com
$ sudo cp $HOME/custom-ee/files/hub-ca.crt /etc/containers/certs.d/private-hub.example.com
```

4.12.4. Uploading the custom execution environment to the private automation hub

Before the new execution environment image can be used for automation jobs, it must be uploaded to the private automation hub.

First, verify that the execution environment image can be seen in the local podman cache:

```
$ podman images --format "table {{.ID}} {{.Repository}} {{.Tag}}"
IMAGE ID     REPOSITORY                    TAG
b38e3299a65e private-hub.example.com/custom-ee        latest
8e38be53b486 private-hub.example.com/ee-minimal-rhel8  latest
```

Then log in to the private automation hub's container registry and push the image to make it available for use with job templates and workflows:

```
$ podman login private-hub.example.com -u admin
Password:
Login Succeeded!
$ podman push private-hub.example.com/custom-ee:latest
```

4.13. 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. Create 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**
nginx_user_http_config	List of nginx configurations for **/etc/nginx/nginx.conf** under the http section. Each element in the list is provided into **http nginx config** as a separate line. Default = empty list
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 a bundle installer.
registry_url	Used for both **[automationcontroller]** and **[automationhub]** groups. Default = **registry.redhat.io**.

Variable	Description
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.
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**. If you do not set**ansible_host**, **inventory_hostname** is used as a last resort. 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 Passwords must be enclosed in quotes when they are provided in plain text in the inventory file.
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.

Variable	Description
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** If any of these are absent, the installation will be halted.
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** to false and the backup/restore process does not then backup or restore **/var/lib/pulp**. Default = **true**.
automationhub_collection_download_count	*Optional* Determines whether download count is displayed on the UI. Default = **false**.

Variable	Description
automationhub_collection_seed_repository	When you run the bundle installer, 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**
automationhub_create_default_collection_signing_service	Set this variable to true to create a collection signing service. 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 container signing service is enabled, you must provide this variable to ensure that containers can be properly signed. **/absolute/path/to/script/that/signs**

Variable	Description
automationhub_create_default_container_signing_service	Set this variable to true to create a container signing service. Default = **false**.
automationhub_disable_hsts	The default installation deploys a TLS enabled Ansible automation hub. Use this variable if you deploy automation hub with *HTTP Strict Transport Security* (HSTS) web-security policy enabled. This variable disables, the HSTS web-security policy mechanism. 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**, this variable 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**.
automationhub_enable_analytics	A Boolean indicating whether to enable pulp analytics for the version of pulpcore used in automation hub in Ansible Automation Platform 2.4. To enable pulp analytics, set **automationhub_enable_analytics** to true. Default = **false**.
automationhub_enable_unauthenticated_collection_access	Set this variable to true to enable unauthorized users to view collections. Default = **false**.
automationhub_enable_unauthenticated_collection_download	Set this variable to true to enable unauthorized users to download collections. Default = **false**.

Variable	Description
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.
automationhub_main_url	The main automation hub URL that clients connect to. For example, https://<load balancer host>. 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. If not specified, the first node in the **[automationhub]** group is used.
automationhub_pg_database	*Required* The database name. Default = **automationhub**.
automationhub_pg_host	Required if not using an internal database. The hostname of the remote PostgreSQL database used by automation hub. Default = **127.0.0.1**.
automationhub_pg_password	The password for the automation hub PostgreSQL database. Use of special characters for **automationhub_pg_password** is limited. The **!**, **#**, **0** and **@** characters are supported. Use of other special characters can cause the setup to fail.
automationhub_pg_port	Required if not using an internal database. Default = 5432.

Variable	Description
automationhub_pg_sslmode	Required. Default = **prefer**.
automationhub_pg_username	Required Default = **automationhub**.
automationhub_require_content_approval	*Optional* Value is **true** 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 they are 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 you run the bundle installer, 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. 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.

Variable	Description
automationhub_ssl_key	*Optional* **/path/to/automationhub.key**. Same as **web_server_ssl_key** but for automation hub UI and API
automationhub_ssl_validate_certs	For Red Hat Ansible Automation Platform 2.2 and later, this value is no longer used. Set value to **true** if automation hub must validate certificates 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.
automationhub_user_headers	List of nginx headers for Ansible automation hub's web server. Each element in the list is provided to the web server's nginx configuration as a separate line. Default = empty list
ee_from_hub_only	When deployed with automation hub the installer pushes execution environment images to automation hub and configures automation controller to pull images from the automation hub registry. To make automation hub the only registry to pull execution environment images from, set this variable to **true**. If set to **false**, execution environment images are also taken directly from Red Hat. Default = **true** when the bundle installer is used.

Variable	Description
generate_automationhub_token	If upgrading from Red Hat Ansible Automation Platform 2.0 or earlier, choose one of the following options: • provide an existing Ansible automation hub token as **automationhub_api_token** • 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. Values available **TLSv1**, **TLSv1.1, `TLSv1.2**, **TLSv1.3** The TLSv1.1 and TLSv1.2 parameters only work when OpenSSL 1.0.1 or higher is used. The TLSv1.3 parameter only works when OpenSSL 1.1.1 or higher is used. If **nginx_tls-protocols = ['TLSv1.3']** only TLSv1.3 is enabled. To set more than one protocol use **nginx_tls_protocols = ['TLSv1.2', 'TLSv.1.3']** Default = **TLSv1.2**.
pulp_db_fields_key	Relative or absolute path to the Fernet symmetric encryption key that 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.

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**.
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.

Variable	Description
sso_host	*Required* Used for Ansible Automation Platform externally managed Red Hat Single Sign-On only. Automation hub requires SSO and SSO administration credentials for authentication. 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**.
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.

Variable	Description
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 must 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**.

For Ansible automation hub to connect to LDAP directly, you must configure the following variables: A list of additional LDAP related variables that can be passed using the **ldap_extra_settings** variable, see the Django reference documentation.

Variable	Description
automationhub_ldap_bind_dn	The name to use when binding to the LDAP server with **automationhub_ldap_bind_password**. Must be set when integrating private automation hub with LDAP, or the installation will fail.
automationhub_ldap_bind_password	*Required* The password to use with **automationhub_ldap_bind_dn**. Must be set when integrating private automation hub LDAP, or the installation will fail.
automationhub_ldap_group_search_base_dn	An LDAP Search object that finds all LDAP groups that users might belong to. If your configuration makes any references to LDAP groups, you must set this variable and **automationhub_ldap_group_type**. Must be set when integrating private automation hub with LDAP, or the installation will fail. Default = **None**

Variable	Description
automationhub_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	Describes the type of group returned by **automationhub_ldap_group_search**. This is set dynamically based on the the values of **automationhub_ldap_group_type_params** and **automationhub_ldap_group_type_class**, otherwise it is the default value coming from django-ldap which is 'None' Default = **django_auth_ldap.config:GroupOfNamesType**
automationhub_ldap_group_type_class	*Optional* The importable path for the django-ldap group type class. 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. Use any URI that is supported by your underlying LDAP libraries. Must be set when integrating private automation hub LDAP, or the installation will fail.

Variable	Description
automationhub_ldap_user_search_base_dn	An LDAP Search object that locates a user in the directory. The filter parameter must contain the placeholder %(user)s for the username. It must return exactly one result for authentication to succeed. Must be set when integrating private automation hub with LDAP, or the installation will fail.
automationhub_ldap_user_search_filter	*Optional* Default = **'(uid=%(user)s)'**
automationhub_ldap_user_search_scope	*Optional* Scope to search for users in an LDAP tree by using the django framework for LDAP authentication. Used for installing automation hub with LDAP. Default = **SUBTREE**

A.3. AUTOMATION CONTROLLER VARIABLES

Variable	Description
admin_password	The admin password used to connect to the automation controller instance. Passwords must be enclosed in quotes when they are provided in plain text in the inventory file.
automation_controller_main_url	The full URL used by Event-Driven Ansible to connect to a controller host. This URL is required if there is no automation controller configured in the inventory file. Format example: **automation_controller_main_url='https://<hostname>'**
admin_username	The username used to identify and create the admin superuser in automation controller.
admin_email	The email address used for the admin user for automation controller.

Variable	Description
nginx_http_port	The nginx HTTP server listens for inbound connections. Default = 80
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 must 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. Values available **TLSv1**, **TLSv1.1, `TLSv1.2**, **TLSv1.3** The TLSv1.1 and TLSv1.2 parameters only work when OpenSSL 1.0.1 or higher is used. The TLSv1.3 parameter only works when OpenSSL 1.1.1 or higher is used. If **nginx_tls-protocols = ['TLSv1.3']** only TLSv1.3 is enabled. To set more than one protocol use **nginx_tls_protocols = ['TLSv1.2', 'TLSv.1.3']** Default = **TLSv1.2**.
nginx_user_headers	List of nginx headers for the automation controller web server. Each element in the list is provided to the web server's nginx configuration as a separate line. Default = empty list
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**.

Variable	Description
node_type	For **[automationcontroller]** group. Two valid **node_types** can be assigned for this group. A **node_type=control** means that the node only runs project and inventory updates, but not regular jobs. A **node_type=hybrid** can 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**.
peers	*Optional* The **peers** variable is used to indicate which nodes a specific host or group connects to. Wherever this variable is defined, an outbound connection to the specific host or group is established. This variable is used to add **tcp-peer** entries in the **receptor.conf** file used for establishing network connections with other nodes. The peers variable can be a comma-separated list of hosts and 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 postgreSQL database. Default = **awx**.
pg_host	The postgreSQL host, which can be an externally managed database.

Variable	Description
pg_password	The password for the postgreSQL database. Use of special characters for **pg_password** is limited. The **!**, **#**, **0** and **@** characters are supported. Use of other special characters can cause the setup 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	Choose one of the two available modes: **prefer** and **verify-full**. Set to **verify-full** for client-side enforced SSL. Default = **prefer**.
pg_username	Your postgreSQL database username. Default = **awx**.
postgres_ssl_cert	Location of the postgreSQL SSL certificate. **/path/to/pgsql_ssl.cert**
postgres_ssl_key	Location of the postgreSQL SSL key. **/path/to/pgsql_ssl.key**
postgres_use_cert	Location of the postgreSQL user certificate. **/path/to/pgsql.crt**
postgres_use_key	Location of the postgreSQL user key. **/path/to/pgsql.key**

Variable	Description
postgres_use_ssl	Use this variable if postgreSQL uses SSL.
postgres_max_connections	Maximum database connections setting to apply if you are using installer-managed postgreSQL. See PostgreSQL database configuration in the automation controller administration guide for help selecting a value. Default for VM-based installations = 200 for a single node and 1024 for a cluster.
receptor_listener_port	Port to use for receptor connection. Default = 27199
supervisor_start_retry_count	When specified, it adds **startretries = <value specified>** to the supervisor config file (/etc/supervisord.d/tower.ini). See program:x Section Values for more information about **startretries**. No default value exists.
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.4. ANSIBLE VARIABLES

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

For more information about variables specific to certain plugins, see the documentation for Ansible.Builtin.

For a list of global configuration options, see Ansible Configuration Settings.

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. Default: 22 for ssh
ansible_user	The user name to use when connecting to the host.
ansible_password	The password 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, 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. Do 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**. Do not change this variable unless /**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.

A.5. EVENT-DRIVEN ANSIBLE CONTROLLER VARIABLES

Variable	Description
automationedacontroller_admin_password	The admin password used by the Event-Driven Ansible controller instance. Passwords must be enclosed in quotes when they are provided in plain text in the inventory file.
automationedacontroller_admin_username	Username used by django to identify and create the admin superuser in Event-Driven Ansible controller. Default = **admin**

Variable	Description
automationedacontroller_admin_email	Email address used by django for the admin user for Event-Driven Ansible controller. Default = **admin@example.com**
automationedacontroller_allowed_hostnames	List of additional addresses to enable for user access to Event-Driven Ansible controller. Default = empty list
automationedacontroller_controller_verify_ssl	Boolean flag used to verify automation controller's web certificates when making calls from Event-Driven Ansible controller. Verified is **true**; not verified is **false**. Default = **false**
automationedacontroller_disable_https	Boolean flag to disable HTTPS Event-Driven Ansible controller. Default = **false**
automationedacontroller_disable_hsts	Boolean flag to disable HSTS Event-Driven Ansible controller. Default = **false**
automationedacontroller_gunicorn_workers	Number of workers for the API served through gunicorn. Default = (# of cores or threads) * 2 + 1
automationedacontroller_max_running_activations	The number of maximum activations running concurrently per node. This is an integer that must be greater than 0. Default = 12
automationedacontroller_nginx_tls_files_remote	Boolean flag to specify whether cert sources are on the remote host (true) or local (false). Default = **false**
automationedacontroller_pg_database	The Postgres database used by Event-Driven Ansible controller. Default = **automtionedacontroller**.

Variable	Description
automationnedacontroller_pg_host	The hostname of the Postgres database used by Event-Driven Ansible controller, which can be an externally managed database.
automationedacontroller_pg_password	The password for the Postgres database used by Event-Driven Ansible controller. Use of special characters for **automationedacontroller_pg_password** is limited. The **!**, **#**, **0** and **@** characters are supported. Use of other special characters can cause the setup to fail.
automationedacontroller_pg_port	The port number of the Postgres database used by Event-Driven Ansible controller. Default = **5432**.
automationedacontroller_pg_username	The username for your Event-Driven Ansible controller Postgres database. Default = **automationedacontroller**.
automationedacontroller_rq_workers	Number of Redis Queue (RQ) workers used by Event-Driven Ansible controller. RQ workers are Python processes that run in the background. Default = (# of cores or threads) * 2 + 1
automationedacontroller_ssl_cert	*Optional* **/root/ssl_certs/eda.*<example>*.com.crt** Same as **automationhub_ssl_cert** but for Event-Driven Ansible controller UI and API.
automationedacontroller_ssl_key	*Optional* **/root/ssl_certs/eda.*<example>*.com.key** Same as **automationhub_server_ssl_key** but for Event-Driven Ansible controller UI and API.
automationedacontroller_user_headers	List of additional nginx headers to add to Event-Driven Ansible controller's nginx configuration. Default = empty list