Red Hat Ansible Automation Platform 2.4

Containerized Ansible Automation Platform installation guide

Containerized Ansible Automation Platform Installation Guide

Last Updated: 2024-07-02

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

Containerized Ansible Automation Platform Installation Guide

Legal Notice

Abstract

This guide helps you to understand the installation requirements and processes behind our new containerized version of Ansible Automation Platform.

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

If you have a suggestion to improve this documentation, or find an error, please contact technical support at https://access.redhat.com to create an issue on the Ansible Automation Platform Jira project using the **docs-product** component.

CHAPTER 1. ANSIBLE AUTOMATION PLATFORM CONTAINERIZED INSTALLATION

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 our new containerized version of Ansible Automation Platform. This initial version is based upon Ansible Automation Platform 2.4 and is being released as a Technical Preview. Please see Technology Preview Features Support Scope to understand what a technical preview entails.

Prerequisites

- A RHEL 9.2 based host. Minimal OS base install is recommended.
- A non-root user for the RHEL host, with sudo or other Ansible supported privilege escalation (sudo recommended). This user is responsible for the installation of containerized Ansible Automation Platform.
- It is recommended setting up an **SSH public key authentication**for the non-root user. For guidelines on setting up an SSH public key authentication for the non-root user, see How to configure SSH public key authentication for passwordless login.
- SSH keys are only required when installing on remote hosts. If doing a self contained local VM based installation, you can use **ansible_connection: local** as per the example which does not require SSH.
- Internet access from the RHEL host if using the default online installation method.

1.1. SYSTEM REQUIREMENTS

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

Memory	16Gb RAM
CPU	4 CPU
Disk space	40Gb
Disk IOPs	1500

1.2. PREPARING THE RHEL HOST FOR CONTAINERIZED INSTALLATION

Procedure

Containerized Ansible Automation Platform runs the component services as podman based containers on top of a RHEL host. The installer takes care of this once the underlying host has been prepared. Use the following instructions for this.

1. Log into your RHEL host as your non-root user.
2. Run **dnf repolist** to validate only the BaseOS and appstream repos are setup and enabled on the host:

```
$ dnf repolist
Updating Subscription Management repositories.
repo id                                  repo name
rhel-9-for-x86_64-appstream-rpms         Red Hat Enterprise Linux 9 for x86_64 - AppStream (RPMs)
rhel-9-for-x86_64-baseos-rpms            Red Hat Enterprise Linux 9 for x86_64 - BaseOS (RPMs)
```

3. Ensure that these repos and only these repos are available to the host OS. If you need to know how to do this use this guide: Chapter 10. Managing custom software repositories Red Hat Enterprise Linux
4. Ensure that the host has DNS configured and can resolve hostnames and IPs using a fully qualified domain name (FQDN). This is essential to ensure services can talk to one another.

Using unbound DNS

To configure unbound DNS refer to Chapter 2. Setting up an unbound DNS server Red Hat Enterprise Linux 9.

Using BIND DNS

To configure DNS using BIND refer to Chapter 1. Setting up and configuring a BIND DNS server Red Hat Enterprise Linux 9.

Optional

To have the installer automatically pick up and apply your Ansible Automation Platform subscription manifest license, use this guide to generate a manifest file which can be downloaded for the installer: Chapter 2. Obtaining a manifest file Red Hat Ansible Automation Platform 2. .

1.3. INSTALLING ANSIBLE-CORE

Procedure

1. Install ansible-core and other tools:

```
sudo dnf install -y ansible-core wget git rsync
```

2. Set a fully qualified hostname:

```
sudo hostnamectl set-hostname your-FQDN-hostname
```

1.4. DOWNLOADING ANSIBLE AUTOMATION PLATFORM

Procedure

1. Download the latest installer tarball from access.redhat.com. This can be done directly within the RHEL host, which saves time.

2. If you have downloaded the tarball and optional manifest zip file onto your laptop, copy them onto your RHEL host.
   Decide where you would like the installer to reside on the filesystem. Installation related files will be created under this location and require at least 10Gb for the initial installation.

3. Unpack the installer tarball into your installation directory, and cd into the unpacked directory.

   a. online installer

   ```
   $ tar xfvz ansible-automation-platform-containerized-setup-2.4-2.tar.gz
   ```

   b. bundled installer

   ```
   $ tar xfvz ansible-automation-platform-containerized-setup-bundle-2.4-2-<arch name>.tar.gz
   ```

1.5. USING POSTINSTALL FEATURE OF CONTAINERIZED ANSIBLE AUTOMATION PLATFORM

Use the experimental postinstaller feature of containerized Ansible Automation Platform to define and load the configuration during the initial installation. This uses a configuration-as-code approach, where you simply define your configuration to be loaded as simple YAML files.

1. To use this optional feature, you need to uncomment the following vars in the inventory file:

   ```
   controller_postinstall=true
   ```

2. The default is false, so you need to enable this to activate the postinstaller. You need a Ansible Automation Platform license for this feature that must reside on the local filesystem so it can be automatically loaded:

   ```
   controller_license_file=/full_path_to/manifest_file.zip
   ```

3. You can pull your configuration-as-code from a Git based repository. To do this, set the following variables to dictate where you pull the content from and where to store it for upload to the Ansible Automation Platform controller:

   ```
   controller_postinstall_repo_url=https://your_cac_scm_repo
   controller_postinstall_dir=/full_path_to_where_you_want_the pulled_content_to_reside
   ```

4. The controller_postinstall_repo_url variable can be used to define the postinstall repository URL which must include authentication information.

   ```
   http(s)://<host>/<repo>.git (public repository without http(s) authentication)
   http(s)://<user>:<password>@<host>:<repo>.git (private repository with http(s) authentication)
   git@<host>:<repo>.git (public/private repository with ssh authentication)
   ```

NOTE

When using ssh based authentication, the installer does not configure anything for you, so you must configure everything on the installer node.

Definition files use the infra certified collections. The controller_configuration collection is preinstalled as part of the installation and uses the installation controller credentials you supply in the inventory file for access into the Ansible Automation Platform controller. You simply need to give the YAML configuration files.

You can setup Ansible Automation Platform configuration attributes such as credentials, LDAP settings, users and teams, organizations, projects, inventories and hosts, job and workflow templates.

The following example shows a sample **your-config.yml** file defining and loading controller job templates. The example demonstrates a simple change to the preloaded demo example provided with an Ansible Automation Platform installation.

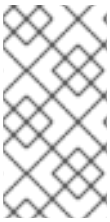
```
/full_path_to_your_configuration_as_code/
├── controller
    └── job_templates.yml
```

```
controller_templates:
 - name: Demo Job Template
   execution_environment: Default execution environment
   instance_groups:
  - default
   inventory: Demo Inventory
```

1.6. INSTALLING CONTAINERIZED ANSIBLE AUTOMATION PLATFORM

Installation of Ansible Automation Platform is controlled with inventory files. Inventory files define the hosts and containers used and created, variables for components, and other information needed to customize the installation.

For convenience an example inventory file is provided, that you can copy and modify to quickly get started.



NOTE

There is no default database choice given in the inventory file. You must follow the instructions in the inventory file to make the appropriate choice between an internally provided postgres, or provide your own externally managed and supported database option.

Edit the inventory file by replacing the **< >** placeholders with your specific variables, and uncommenting any lines specific to your needs.

```
# This is the AAP installer inventory file
# Please consult the docs if you're unsure what to add
# For all optional variables please consult the included README.md

# This section is for your AAP Controller host(s)
# -----------------------------------------------
```

```
[automationcontroller]
fqdn_of_your_rhel_host ansible_connection=local

# This section is for your AAP Automation Hub host(s)
# -----------------------------------------------------
[automationhub]
fqdn_of_your_rhel_host ansible_connection=local

# This section is for your AAP EDA Controller host(s)
# -----------------------------------------------------
[automationeda]
fqdn_of_your_rhel_host ansible_connection=local

# This section is for your AAP Execution host(s)
# -----------------------------------------------
#[execution_nodes]
#fqdn_of_your_rhel_host

# This section is for the AAP database(s)
# ---------------------------------------
# Uncomment the lines below and amend appropriately if you want AAP to install and manage the
postgres databases
# Leave commented out if you intend to use your own external database and just set appropriate
_pg_hosts vars
# see mandatory sections under each AAP component
#[database]
#fqdn_of_your_rhel_host ansible_connection=local

[all:vars]

# Common variables needed for installation
# -----------------------------------------
postgresql_admin_username=postgres
postgresql_admin_password=<set your own>
# If using the online (non-bundled) installer, you need to set RHN registry credentials
registry_username=<your RHN username>
registry_password=<your RHN password>
# If using the bundled installer, you need to alter defaults by using:
#bundle_install=true
# The bundle directory must include /bundle in the path
#bundle_dir=<full path to the bundle directory>
# To add more decision environment images you need to set the de_extra_images variable
#de_extra_images=[{'name': 'Custom decision environment', 'image':
'<registry>/<namespace>/<image>:<tag>'}]
# To add more execution environment images you need to set the ee_extra_images variable
#ee_extra_images=[{'name': 'Custom execution environment', 'image':
'<registry>/<namespace>/<image>:<tag>'}]
# To use custom TLS CA certificate/key you need to set these variables
#ca_tls_cert=<full path to your TLS CA certificate file>
#ca_tls_key=<full path to your TLS CA key file>

# AAP Database - optional
# -------------------------
# To use custom TLS certificate/key you need to set these variables
#postgresql_tls_cert=<full path to your TLS certificate file>
#postgresql_tls_key=<full path to your TLS key file>
```

```
# AAP Controller - mandatory
# -------------------------
controller_admin_password=<set your own>
controller_pg_host=fqdn_of_your_rhel_host
controller_pg_password=<set your own>

# AAP Controller - optional
# -------------------------
# To use the postinstall feature you need to set these variables
#controller_postinstall=true
#controller_license_file=<full path to your manifest .zip file>
#controller_postinstall_dir=<full path to your config-as-code directory>
# When using config-as-code in a git repository
#controller_postinstall_repo_url=<url to your config-as-code git repository>
#controller_postinstall_repo_ref=main
# To use custom TLS certificate/key you need to set these variables
#controller_tls_cert=<full path to your TLS certificate file>
#controller_tls_key=<full path to your TLS key file>

# AAP Automation Hub - mandatory
# -----------------------------
hub_admin_password=<set your own>
hub_pg_host=fqdn_of_your_rhel_host
hub_pg_password=<set your own>

# AAP Automation Hub - optional
# -----------------------------
# To use the postinstall feature you need to set these variables
#hub_postinstall=true
#hub_postinstall_dir=<full path to your config-as-code directory>
# When using config-as-code in a git repository
#hub_postinstall_repo_url=<url to your config-as-code git repository>
#hub_postinstall_repo_ref=main
# To customize the number of worker containers
#hub_workers=2
# To use the collection signing feature you need to set these variables
#hub_collection_signing=true
#hub_collection_signing_key=<full path to your gpg key file>
# To use the container signing feature you need to set these variables
#hub_container_signing=true
#hub_container_signing_key=<full path to your gpg key file>
# To use custom TLS certificate/key you need to set these variables
#hub_tls_cert=<full path to your TLS certificate file>
#hub_tls_key=<full path to your TLS key file>

# AAP EDA Controller - mandatory
# -----------------------------
eda_admin_password=<set your own>
eda_pg_host=fqdn_of_your_rhel_host
eda_pg_password=<set your own>

# AAP EDA Controller - optional
# -----------------------------
# When using an external controller node unmanaged by the installer.
#controller_main_url=https://fqdn_of_your_rhel_host
```

```
# To customize the number of default/activation worker containers
#eda_workers=2
#eda_activation_workers=2
# To use custom TLS certificate/key you need to set these variables
#eda_tls_cert=<full path to your TLS certificate file>
#eda_tls_key=<full path to your TLS key file>

# AAP Execution Nodes - optional
# ----------------------------
#receptor_port=27199
#receptor_protocol=tcp
# To use custom TLS certificate/key you need to set these variables
#receptor_tls_cert=<full path to your TLS certificate file>
#receptor_tls_key=<full path to your TLS key file>
# To use custom RSA key pair you need to set these variables
#receptor_signing_private_key=<full path to your RSA private key file>
#receptor_signing_public_key=<full path to your RSA public key file>
```

Use the following command to install containerized Ansible Automation Platform:

```
ansible-playbook -i inventory ansible.containerized_installer.install
```

NOTE

If your privilege escalation requires a password to be entered, append *-K* to the command line. You will then be prompted for the *BECOME* password.

You can use increasing verbosity, up to 4 v's (-vvvv) to see the details of the installation process.

NOTE

This can significantly increase installation time, so it is recommended that you use it only as needed or requested by Red Hat support.

1.7. ACCESSING AUTOMATION CONTROLLER, AUTOMATION HUB, AND EVENT-DRIVEN ANSIBLE CONTROLLER

After the installation completes, these are the default protocol and ports used:

- http/https protocol
- Ports 8080/8443 for automation controller
- Ports 8081/8444 for automation hub
- Ports 8082/8445 for Event-Driven Ansible controller

These can be changed. Consult the **README.md** for further details. It is recommended that you leave the defaults unless you need to change them due to port conflicts or other factors.

Accessing automation controller UI

The automation controller UI is available by default at:

```
https://<your_rhel_host>:8443
```

Log in as the admin user with the password you created for **controller_admin_password**.

If you supplied the license manifest as part of the installation, the Ansible Automation Platform dashboard is displayed. If you did not supply a license file, the **Subscription** screen is displayed where you must supply your license details. This is documented here: Chapter 1. Activating Red Hat Ansible Automation Platform.

Accessing automation hub UI

The automation hub UI is available by default at:

```
https://<hub node>:8444
```

Log in as the admin user with the password you created for **hub_admin_password**.

Accessing Event-Driven Ansible UI

The Event-Driven Ansible UI is available by default at:

```
https://<eda node>:8445
```

Log in as the admin user with the password you created for **eda_admin_password**.

1.8. USING CUSTOM TLS CERTIFICATES

By default, the installer generates TLS certificates and keys for all services which are signed by a custom Certificate Authority (CA). You can provide a custom TLS certificate/key for each service. If that certificate is signed by a custom CA, you must provide the CA TLS certificate and key.

- Certificate Authority

```
ca_tls_cert=/full/path/to/tls/certificate
ca_tls_key=/full/path/to/tls/key
```

- Automation Controller

```
controller_tls_cert=/full/path/to/tls/certificate
controller_tls_key=/full/path/to/tls/key
```

- Automation Hub

```
hub_tls_cert=/full/path/to/tls/certificate
hub_tls_key=/full/path/to/tls/key
```

- Automation EDA

```
eda_tls_cert=/full/path/to/tls/certificate
eda_tls_key=/full/path/to/tls/key
```

- Postgresql

```
postgresql_tls_cert=/full/path/to/tls/certificate
postgresql_tls_key=/full/path/to/tls/key
```

- Receptor

```
receptor_tls_cert=/full/path/to/tls/certificate
receptor_tls_key=/full/path/to/tls/key
```

1.9. USING CUSTOM RECEPTOR SIGNING KEYS

Receptor signing is now enabled by default unless **receptor_disable_signing=true** is set, and a RSA key pair (public/private) is generated by the installer. However, you can provide custom RSA public/private keys by setting the path variable.

```
receptor_signing_private_key=/full/path/to/private/key
receptor_signing_public_key=/full/path/to/public/key
```

1.10. ENABLING AUTOMATION HUB COLLECTION AND CONTAINER SIGNING

Automation hub allows you to sign Ansible collections and container images. This feature is not enabled by default, and you must provide the GPG key.

```
hub_collection_signing=true
hub_collection_signing_key=/full/path/to/collections/gpg/key
hub_container_signing=true
hub_container_signing_key=/full/path/to/containers/gpg/key
```

When the GPG key is protected by a passphrase, you must provide the passphrase.

```
hub_collection_signing_pass=<collections gpg key passphrase>
hub_container_signing_pass=<containers gpg key passphrase>
```

1.11. ADDING EXECUTION NODES

The containerized installer can deploy remote execution nodes. This is handled by the execution_nodes group in the ansible inventory file.

```
[execution_nodes]
fqdn_of_your_execution_host
```

An execution node is by default configured as an execution type running on port 27199 (TCP). This can be changed by the following variables:

- receptor_port=27199
- receptor_protocol=tcp
- receptor_type=hop

Receptor type value can be either execution or hop, while the protocol is either TCP or UDP. By default, the nodes in the **execution_nodes** group will be added as peers for the controller node. However, you can change the peers configuration by using the **receptor_peers** variable.

```
[execution_nodes]
fqdn_of_your_execution_host
fqdn_of_your_hop_host receptor_type=hop receptor_peers='["fqdn_of_your_execution_host"]'
```

1.12. UNINSTALLING CONTAINERIZED ANSIBLE AUTOMATION PLATFORM

To uninstall a containerized deployment, execute the **uninstall.yml** playbook.

```
$ ansible-playbook -i inventory ansible.containerized_installer.uninstall
```

This will stop all systemd units and containers and then delete all resources used by the containerized installer such as:

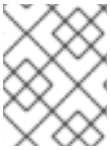
- config and data directories/files
- systemd unit files
- podman containers and images
- RPM packages

To keep container images, you can set the **container_keep_images** variable to true.

```
$ ansible-playbook -i inventory ansible.containerized_installer.uninstall -e
container_keep_images=true
```

To keep postgresql databases, you can set the **postgresql_keep_databases** variable to true.

```
$ ansible-playbook -i </path/to/inventory> ansible.containerized_installer.uninstall -e
postgresql_keep_databases=true
```



NOTE

You will have to use the same django secret key values rather than the auto-generated ones.

APPENDIX A. TROUBLESHOOTING CONTAINERIZED ANSIBLE AUTOMATION PLATFORM

Use this information to troubleshoot your containerized Ansible Automation Platform installation.

A.1. TROUBLESHOOTING CONTAINERIZED ANSIBLE AUTOMATION PLATFORM INSTALLATION

The installation takes a long time, or has errors, what should I check?

1. Ensure your system meets the minimum requirements as outlined in the installation guide. Items such as improper storage choices and high latency when distributing across many hosts will all have a significant impact.

2. Check the installation log file located by default at **./aap_install.log** unless otherwise changed within the local installer **ansible.cfg**.

3. Enable task profiling callbacks on an ad hoc basis to give an overview of where the installation program spends the most time. To do this, use the local **ansible.cfg** file. Add a callback line such as this under the **[defaults]** section:

```
$ cat ansible.cfg
[defaults]
callbacks_enabled = ansible.posix.profile_tasks
```

Automation controller returns an error of 413

This error is due to **manifest.zip** license files that are larger than the **nginx_client_max_body_size** setting. If this error occurs, you will need to change the installation inventory file to include the following variables:

```
nginx_disable_hsts: false
nginx_http_port: 8081
nginx_https_port: 8444
nginx_client_max_body_size: 20m
nginx_user_headers: []
```

The current default setting of **20m** should be enough to avoid this issue.

The installation failed with a "502 Bad Gateway" when going to the controller UI.

This error can occur and manifest itself in the installation application output as:

```
TASK [ansible.containerized_installer.automationcontroller : Wait for the Controller API to te ready]
**************************************************
fatal: [daap1.lan]: FAILED! => {"changed": false, "connection": "close", "content_length": "150",
"content_type": "text/html", "date": "Fri, 29 Sep 2023 09:42:32 GMT", "elapsed": 0, "msg": "Status
code was 502 and not [200]: HTTP Error 502: Bad Gateway", "redirected": false, "server": "nginx",
"status": 502, "url": "https://daap1.lan:443/api/v2/ping/"}
```

- Check if you have an **automation-controller-web** container running and a systemd service.

NOTE

This is used at the regular unprivileged user not system wide level. If you have used **su** to switch to the user running the containers, you must set your **XDG_RUNTIME_DIR** environment variable to the correct value to be able to interact with the user **systemctl** units.

```
export XDG_RUNTIME_DIR="/run/user/$UID"
```

```
podman ps | grep web
systemctl --user | grep web
```

No output indicates a problem.

1. Try restarting the **automation-controller-web** service:

```
systemctl start automation-controller-web.service --user
systemctl --user | grep web
systemctl status automation-controller-web.service --user
```

```
Sep 29 10:55:16 daap1.lan automation-controller-web[29875]: nginx: [emerg] bind() to
0.0.0.0:443 failed (98: Address already in use)
Sep 29 10:55:16 daap1.lan automation-controller-web[29875]: nginx: [emerg] bind() to
0.0.0.0:80 failed (98: Address already in use)
```

The output indicates that the port is already, or still, in use by another service. In this case **nginx**.

2. Run:

```
sudo pkill nginx
```

3. Restart and status check the web service again.

Normal service output should look similar to the following, and should still be running:

```
Sep 29 10:59:26 daap1.lan automation-controller-web[30274]: WSGI app 0 (mountpoint='/') ready in
3 seconds on interpreter 0x1a458c10 pid: 17 (default app)
Sep 29 10:59:26 daap1.lan automation-controller-web[30274]: WSGI app 0 (mountpoint='/') ready in
3 seconds on interpreter 0x1a458c10 pid: 20 (default app)
Sep 29 10:59:27 daap1.lan automation-controller-web[30274]: 2023-09-29 09:59:27,043 INFO     [-]
daphne.cli Starting server at tcp:port=8051:interface=127.0.>
Sep 29 10:59:27 daap1.lan automation-controller-web[30274]: 2023-09-29 09:59:27,043 INFO
Starting server at tcp:port=8051:interface=127.0.0.1
Sep 29 10:59:27 daap1.lan automation-controller-web[30274]: 2023-09-29 09:59:27,048 INFO     [-]
daphne.server HTTP/2 support not enabled (install the http2 >
Sep 29 10:59:27 daap1.lan automation-controller-web[30274]: 2023-09-29 09:59:27,048 INFO
HTTP/2 support not enabled (install the http2 and tls Twisted ex>
Sep 29 10:59:27 daap1.lan automation-controller-web[30274]: 2023-09-29 09:59:27,049 INFO     [-]
daphne.server Configuring endpoint tcp:port=8051:interface=1>
Sep 29 10:59:27 daap1.lan automation-controller-web[30274]: 2023-09-29 09:59:27,049 INFO
Configuring endpoint tcp:port=8051:interface=127.0.0.1
Sep 29 10:59:27 daap1.lan automation-controller-web[30274]: 2023-09-29 09:59:27,051 INFO     [-]
daphne.server Listening on TCP address 127.0.0.1:8051
Sep 29 10:59:27 daap1.lan automation-controller-web[30274]: 2023-09-29 09:59:27,051 INFO
```

```
Listening on TCP address 127.0.0.1:8051
Sep 29 10:59:54 daap1.lan automation-controller-web[30274]: 2023-09-29 09:59:54,139 INFO
success: nginx entered RUNNING state, process has stayed up for > th>
Sep 29 10:59:54 daap1.lan automation-controller-web[30274]: 2023-09-29 09:59:54,139 INFO
success: nginx entered RUNNING state, process has stayed up for > th>
Sep 29 10:59:54 daap1.lan automation-controller-web[30274]: 2023-09-29 09:59:54,139 INFO
success: uwsgi entered RUNNING state, process has stayed up for > th>
Sep 29 10:59:54 daap1.lan automation-controller-web[30274]: 2023-09-29 09:59:54,139 INFO
success: uwsgi entered RUNNING state, process has stayed up for > th>
Sep 29 10:59:54 daap1.lan automation-controller-web[30274]: 2023-09-29 09:59:54,139 INFO
success: daphne entered RUNNING state, process has stayed up for > t>
Sep 29 10:59:54 daap1.lan automation-controller-web[30274]: 2023-09-29 09:59:54,139 INFO
success: daphne entered RUNNING state, process has stayed up for > t>
Sep 29 10:59:54 daap1.lan automation-controller-web[30274]: 2023-09-29 09:59:54,139 INFO
success: ws-heartbeat entered RUNNING state, process has stayed up f>
Sep 29 10:59:54 daap1.lan automation-controller-web[30274]: 2023-09-29 09:59:54,139 INFO
success: ws-heartbeat entered RUNNING state, process has stayed up f>
Sep 29 10:59:54 daap1.lan automation-controller-web[30274]: 2023-09-29 09:59:54,139 INFO
success: cache-clear entered RUNNING state, process has stayed up fo>
Sep 29 10:59:54 daap1.lan automation-controller-web[30274]: 2023-09-29 09:59:54,139 INFO
success: cache-clear entered RUNNING state, process has stayed up
```

You can run the installation program again to ensure everything installs as expected.

When attempting to install containerized Ansible Automation Platform in Amazon Web Services you receive output that there is no space left on device

```
TASK [ansible.containerized_installer.automationcontroller : Create the receptor container]
**************************************************
fatal: [ec2-13-48-25-168.eu-north-1.compute.amazonaws.com]: FAILED! => {"changed": false, "msg":
"Can't create container receptor", "stderr": "Error: creating container storage: creating an ID-mapped
copy of layer \"98955f43cc908bd50ff43585fec2c7dd9445eaf05eecd1e3144f93ffc00ed4ba\": error
during chown: storage-chown-by-maps: lchown usr/local/lib/python3.9/site-
packages/azure/mgmt/network/v2019_11_01/operations/__pycache__/_available_service_aliases_oper
ations.cpython-39.pyc: no space left on device: exit status 1\n", "stderr_lines": ["Error: creating
container storage: creating an ID-mapped copy of layer
\"98955f43cc908bd50ff43585fec2c7dd9445eaf05eecd1e3144f93ffc00ed4ba\": error during chown:
storage-chown-by-maps: lchown usr/local/lib/python3.9/site-
packages/azure/mgmt/network/v2019_11_01/operations/__pycache__/_available_service_aliases_oper
ations.cpython-39.pyc: no space left on device: exit status 1"], "stdout": "", "stdout_lines": []}
```

If you are installing a **/home** filesystem into a default Amazon Web Services marketplace RHEL instance, it might be too small since **/home** is part of the root / filesystem. You will need to make more space available. The documentation specifies a minimum of 40GB for a single-node deployment of containerized Ansible Automation Platform.

A.2. TROUBLESHOOTING CONTAINERIZED ANSIBLE AUTOMATION PLATFORM CONFIGURATION

Sometimes the post install for seeding my Ansible Automation Platform content errors out. This could manifest itself as output similar to this:

```
TASK [infra.controller_configuration.projects : Configure Controller Projects | Wait for finish the
```

```
projects creation] ************************************
Friday 29 September 2023  11:02:32 +0100 (0:00:00.443)       0:00:53.521 ******
FAILED - RETRYING: [daap1.lan]: Configure Controller Projects | Wait for finish the projects creation
(1 retries left).
failed: [daap1.lan] (item={'failed': 0, 'started': 1, 'finished': 0, 'ansible_job_id': '536962174348.33944',
'results_file': '/home/aap/.ansible_async/536962174348.33944', 'changed': False,
'__controller_project_item': {'name': 'AAP Config-As-Code Examples', 'organization': 'Default',
'scm_branch': 'main', 'scm_clean': 'no', 'scm_delete_on_update': 'no', 'scm_type': 'git',
'scm_update_on_launch': 'no', 'scm_url': 'https://github.com/user/repo.git'}, 'ansible_loop_var':
'__controller_project_item'}) => {"__projects_job_async_results_item": {"__controller_project_item":
{"name": "AAP Config-As-Code Examples", "organization": "Default", "scm_branch": "main",
"scm_clean": "no", "scm_delete_on_update": "no", "scm_type": "git", "scm_update_on_launch": "no",
"scm_url": "https://github.com/user/repo.git"}, "ansible_job_id": "536962174348.33944",
"ansible_loop_var": "__controller_project_item", "changed": false, "failed": 0, "finished": 0,
"results_file": "/home/aap/.ansible_async/536962174348.33944", "started": 1}, "ansible_job_id":
"536962174348.33944", "ansible_loop_var": "__projects_job_async_results_item", "attempts": 30,
"changed": false, "finished": 0, "results_file": "/home/aap/.ansible_async/536962174348.33944",
"started": 1, "stderr": "", "stderr_lines": [], "stdout": "", "stdout_lines": []}
```

The **infra.controller_configuration.dispatch** role uses an asynchronous loop with 30 retries to apply each configuration type, and the default delay between retries is 1 second. If the configuration is large, this might not be enough time to apply everything before the last retry occurs.

Increase the retry delay by setting the **controller_configuration_async_delay** variable to something other than 1 second. For example, setting it to 2 seconds doubles the retry time. The place to do this would be in the repository where the controller configuration is defined. It could also be added to the **[all:vars]** section of the installation program inventory file.

A few instances have shown that no additional modification is required, and re-running the installation program again worked.

A.3. CONTAINERIZED ANSIBLE AUTOMATION PLATFORM REFERENCE

Can you provide details of the architecture for the Ansible Automation Platform containerized design?

We use as much of the underlying native RHEL technology as possible. For the container runtime and management of services we use Podman. Many Podman services and commands are used to show and investigate the solution.

For instance, use **podman ps**, and **podman images** to see some of the foundational and running pieces:

```
[aap@daap1 aap]$ podman ps
CONTAINER ID  IMAGE                                                                COMMAND               CREATED
STATUS         PORTS       NAMES
88ed40495117  registry.redhat.io/rhel8/postgresql-13:latest                        run-postgresql        48
minutes ago  Up 47 minutes              postgresql
8f55ba612f04  registry.redhat.io/rhel8/redis-6:latest                             run-redis             47
minutes ago  Up 47 minutes              redis
56c40445c590  registry.redhat.io/ansible-automation-platform-24/ee-supported-rhel8:latest
/usr/bin/receptor...  47 minutes ago  Up 47 minutes              receptor
f346f05d56ee  registry.redhat.io/ansible-automation-platform-24/controller-rhel8:latest
```

```
/usr/bin/launch_a...  47 minutes ago  Up 45 minutes              automation-controller-rsyslog
26e3221963e3  registry.redhat.io/ansible-automation-platform-24/controller-rhel8:latest
/usr/bin/launch_a...  46 minutes ago  Up 45 minutes              automation-controller-task
c7ac92a1e8a1  registry.redhat.io/ansible-automation-platform-24/controller-rhel8:latest
/usr/bin/launch_a...  46 minutes ago  Up 28 minutes              automation-controller-web

[aap@daap1 aap]$ podman images
REPOSITORY                                                TAG         IMAGE ID      CREATED      SIZE
registry.redhat.io/ansible-automation-platform-24/ee-supported-rhel8  latest      b497bdbee59e  10
days ago  3.16 GB
registry.redhat.io/ansible-automation-platform-24/controller-rhel8    latest      ed8ebb1c1baa  10 days
ago  1.48 GB
registry.redhat.io/rhel8/redis-6                          latest      78905519bb05  2 weeks ago  357 MB
registry.redhat.io/rhel8/postgresql-13                      latest      9b65bc3d0413  2 weeks ago  765
MB
[aap@daap1 aap]$
```

Containerized Ansible Automation Platform runs as rootless containers for maximum out-of-the-box security. This means you can install containerized Ansible Automation Platform by using any local unprivileged user account. Privilege escalation is only needed for certain root level tasks, and by default is not needed to use root directly.

Once installed, you will notice certain items have populate on the filesystem where the installation program is run (the underlying RHEL host).

```
[aap@daap1 aap]$ tree -L 1
.
├── aap_install.log
├── ansible.cfg
├── collections
├── galaxy.yml
├── inventory
├── LICENSE
├── meta
├── playbooks
├── plugins
├── README.md
├── requirements.yml
├── roles
```

Other containerized services that make use of things such as Podman volumes, reside under the installation root directory used. Here are some examples for further reference:

The containers directory contains some of the Podman specifics used and installed for the execution plane:

```
containers/
├── podman
├── storage
│   ├── defaultNetworkBackend
│   ├── libpod
│   ├── networks
│   ├── overlay
│   ├── overlay-containers
│   ├── overlay-images
```

```
│   ├── overlay-layers
│   ├── storage.lock
│   └── userns.lock
└── storage.conf
```

The controller directory has some of the installed configuration and runtime data points:

```
controller/
├── data
│   ├── job_execution
│   ├── projects
│   └── rsyslog
├── etc
│   ├── conf.d
│   ├── launch_awx_task.sh
│   ├── settings.py
│   ├── tower.cert
│   └── tower.key
├── nginx
│   └── etc
├── rsyslog
│   └── run
└── supervisor
    └── run
```

The receptor directory has the automation mesh configuration:

```
receptor/
├── etc
│   └── receptor.conf
└── run
    ├── receptor.sock
    └── receptor.sock.lock
```

After installation, you will also find other pieces in the local users home directory such as the **.cache** directory:

```
.cache/
├── containers
│   └── short-name-aliases.conf.lock
└── rhsm
    └── rhsm.log
```

As we run by default in the most secure manner, such as rootless Podman, we can also use other services such as running **systemd** as non-privileged users. Under **systemd** you can see some of the component service controls available:

The **.config** directory:

```
.config/
├── cni
│   └── net.d
│       └── cni.lock
├── containers
│   ├── auth.json
```

```
│   └── containers.conf
└── systemd
    └── user
        ├── automation-controller-rsyslog.service
        ├── automation-controller-task.service
        ├── automation-controller-web.service
        ├── default.target.wants
        ├── podman.service.d
        ├── postgresql.service
        ├── receptor.service
        ├── redis.service
        └── sockets.target.wants
```

This is specific to Podman and conforms to the Open Container Initiative (OCI) specifications. Whereas Podman run as the root user would use **/var/lib/containers** by default, for standard users the hierarchy under **$HOME/.local** is used.

The **.local** directory:

```
.local/
└── share
    └── containers
        ├── cache
        ├── podman
        └── storage
```

As an example `.local/storage/volumes` contains what the output from `podman volume ls` provides:

```
[aap@daap1 containers]$ podman volume ls
DRIVER      VOLUME NAME
local       d73d3fe63a957bee04b4853fd38c39bf37c321d14fdab9ee3c9df03645135788
local       postgresql
local       redis_data
local       redis_etc
local       redis_run
```

We isolate the execution plane from the control plane main services (PostgreSQL, Redis, automation controller, receptor, automation hub and Event-Driven Ansible).

Control plane services run with the standard Podman configuration (**~/.local/share/containers/storage**).

Execution plane services use a dedicated configuration or storage (**~/aap/containers/storage**) to avoid execution plane containers to be able to interact with the control plane.

How can I see host resource utilization statistics?

- Run:

```
$ podman container stats -a
```

```
podman container stats -a
ID          NAME                        CPU %       MEM USAGE / LIMIT  MEM %       NET IO      BLOCK
IO    PIDS        CPU TIME    AVG CPU %
0d5d8eb93c18  automation-controller-web      0.23%       959.1MB / 3.761GB  25.50%      0B / 0B
```

```
0B / 0B     16          20.885142s  1.19%
3429d559836d  automation-controller-rsyslog  0.07%       144.5MB / 3.761GB  3.84%       0B / 0B
0B / 0B     6           4.099565s   0.23%
448d0bae0942  automation-controller-task     1.51%       633.1MB / 3.761GB  16.83%      0B / 0B     0B
/ 0B     33          34.285272s  1.93%
7f140e65b57e  receptor                      0.01%       5.923MB / 3.761GB  0.16%       0B / 0B     0B / 0B
7           1.010613s   0.06%
c1458367ca9c  redis                         0.48%       10.52MB / 3.761GB  0.28%       0B / 0B     0B / 0B     5
9.074042s   0.47%
ef712cc2dc89  postgresql                    0.09%       21.88MB / 3.761GB  0.58%       0B / 0B     0B / 0B
21          15.571059s  0.80%
```

The previous is an example of a Dell sold and offered containerized Ansible Automation Platform solution (DAAP) install and utilizes ~1.8Gb RAM.

How much storage is used and where?

As we run rootless Podman the container volume storage is under the local user at **$HOME/.local/share/containers/storage/volumes**.

1. To view the details of each volume run:

```
$ podman volume ls
```

2. Then run:

```
$ podman volume inspect <volume_name>
```

Here is an example:

```
$ podman volume inspect postgresql
[
    {
        "Name": "postgresql",
        "Driver": "local",
        "Mountpoint": "/home/aap/.local/share/containers/storage/volumes/postgresql/_data",
        "CreatedAt": "2024-01-08T23:39:24.983964686Z",
        "Labels": {},
        "Scope": "local",
        "Options": {},
        "MountCount": 0,
        "NeedsCopyUp": true
    }
]
```

Several files created by the installation program are located in **$HOME/aap/** and bind-mounted into various running containers.

1. To view the mounts associated with a container run:

```
$ podman ps --format "{{.ID}}\t{{.Command}}\t{{.Names}}"
```

```
Example:
$ podman ps --format "{{.ID}}\t{{.Command}}\t{{.Names}}"
```

```
89e779b81b83 run-postgresql postgresql
4c33cc77ef7d run-redis redis
3d8a028d892d /usr/bin/receptor... receptor
09821701645c /usr/bin/launch_a... automation-controller-rsyslog
a2ddb5cac71b /usr/bin/launch_a... automation-controller-task
fa0029a3b003 /usr/bin/launch_a... automation-controller-web
20f192534691 gunicorn --bind 1... automation-eda-api
f49804c7e6cb daphne -b 127.0.0... automation-eda-daphne
d340b9c1cb74 /bin/sh -c nginx ... automation-eda-web
111f47de5205 aap-eda-manage rq... automation-eda-worker-1
171fcb1785af aap-eda-manage rq... automation-eda-worker-2
049d10555b51 aap-eda-manage rq... automation-eda-activation-worker-1
7a78a41a8425 aap-eda-manage rq... automation-eda-activation-worker-2
da9afa8ef5e2 aap-eda-manage sc... automation-eda-scheduler
8a2958be9baf gunicorn --name p... automation-hub-api
0a8b57581749 gunicorn --name p... automation-hub-content
68005b987498 nginx -g daemon o... automation-hub-web
cb07af77f89f pulpcore-worker automation-hub-worker-1
a3ba05136446 pulpcore-worker automation-hub-worker-2
```

2. Then run:

```
$ podman inspect <container_name> | jq -r .[].Mounts[].Source
```

```
Example:
/home/aap/.local/share/containers/storage/volumes/receptor_run/_data
/home/aap/.local/share/containers/storage/volumes/redis_run/_data
/home/aap/aap/controller/data/rsyslog
/home/aap/aap/controller/etc/tower.key
/home/aap/aap/controller/etc/conf.d/callback_receiver_workers.py
/home/aap/aap/controller/data/job_execution
/home/aap/aap/controller/nginx/etc/controller.conf
/home/aap/aap/controller/etc/conf.d/subscription_usage_model.py
/home/aap/aap/controller/etc/conf.d/cluster_host_id.py
/home/aap/aap/controller/etc/conf.d/insights.py
/home/aap/aap/controller/rsyslog/run
/home/aap/aap/controller/data/projects
/home/aap/aap/controller/etc/settings.py
/home/aap/aap/receptor/etc/receptor.conf
/home/aap/aap/controller/etc/conf.d/execution_environments.py
/home/aap/aap/tls/extracted
/home/aap/aap/controller/supervisor/run
/home/aap/aap/controller/etc/uwsgi.ini
/home/aap/aap/controller/etc/conf.d/container_groups.py
/home/aap/aap/controller/etc/launch_awx_task.sh
/home/aap/aap/controller/etc/tower.cert
```

3. If the **jq** RPM is not installed, install with:

```
$ sudo dnf -y install jq
```