



Red Hat Ansible Automation Platform 2.5

Creating and using execution environments

Create and use execution environment containers

Red Hat Ansible Automation Platform 2.5 Creating and using execution environments

Create and use execution environment containers

Legal Notice

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This guide shows how to create consistent and reproducible automation execution environments for your Red Hat Ansible Automation Platform. This document includes content from the upstream docs.ansible.com documentation, which is covered by the Apache 2.0 license.

Table of Contents

PREFACE	4
PROVIDING FEEDBACK ON RED HAT DOCUMENTATION	5
CHAPTER 1. INTRODUCTION TO AUTOMATION EXECUTION ENVIRONMENTS	6
1.1. ABOUT AUTOMATION EXECUTION ENVIRONMENTS	6
1.1.1. Why use automation execution environments?	6
CHAPTER 2. USING ANSIBLE BUILDER	7
2.1. WHY USE ANSIBLE BUILDER?	7
2.2. INSTALLING ANSIBLE BUILDER	7
2.3. BUILDING AN EXECUTION ENVIRONMENT IN A DISCONNECTED ENVIRONMENT	7
2.4. BUILDING A DEFINITION FILE	8
2.5. BUILDING THE AUTOMATION EXECUTION ENVIRONMENT IMAGE	9
2.6. BREAKDOWN OF DEFINITION FILE CONTENT	10
2.6.1. Build args and base image	10
2.6.1.1. Galaxy	11
2.6.1.2. Python	11
2.6.1.3. System	11
2.6.2. Images	12
2.6.3. Additional build files	12
2.6.4. Additional custom build steps	13
2.6.5. Additional resources	14
2.7. OPTIONAL BUILD COMMAND ARGUMENTS	14
2.8. CONTAINERFILE	15
2.9. CREATING A CONTAINERFILE WITHOUT BUILDING AN IMAGE	15
CHAPTER 3. COMMON AUTOMATION EXECUTION ENVIRONMENT SCENARIOS	16
3.1. UPDATING THE AUTOMATION HUB CA CERTIFICATE	16
3.2. USING AUTOMATION HUB AUTHENTICATION DETAILS WHEN BUILDING AUTOMATION EXECUTION ENVIRONMENTS	16
3.3. ADDITIONAL RESOURCES	17
CHAPTER 4. PUBLISHING AN AUTOMATION EXECUTION ENVIRONMENT	18
4.1. CUSTOMIZING AN EXISTING AUTOMATION EXECUTION ENVIRONMENTS IMAGE	18
4.2. ADDITIONAL RESOURCES (OR NEXT STEPS)	20
CHAPTER 5. POPULATING YOUR PRIVATE AUTOMATION HUB CONTAINER REGISTRY	21
5.1. PULLING EXECUTION ENVIRONMENTS FOR USE IN AUTOMATION HUB	21
5.2. TAGGING EXECUTION ENVIRONMENTS FOR USE IN AUTOMATION HUB	23
5.3. PUSHING AN EXECUTION ENVIRONMENT TO PRIVATE AUTOMATION HUB	23
CHAPTER 6. SETTING UP YOUR CONTAINER REPOSITORY	25
6.1. PREREQUISITES TO SETTING UP YOUR REMOTE REGISTRY	25
6.2. ADDING A README TO YOUR CONTAINER REPOSITORY	25
6.3. PROVIDING ACCESS TO YOUR AUTOMATION EXECUTION ENVIRONMENTSS	25
6.4. TAGGING CONTAINER IMAGES	26
6.5. CREATING A CREDENTIAL	26
CHAPTER 7. PULLING IMAGES FROM A CONTAINER REPOSITORY	28
7.1. PULLING AN IMAGE	28
7.2. SYNCING IMAGES FROM A CONTAINER REPOSITORY	28
PART I. OPEN SOURCE LICENSE	30

APPENDIX A. AUTOMATION EXECUTION ENVIRONMENTS PRECEDENCE 33

PREFACE

Use execution environment builder to create consistent and reproducible containers for your Red Hat Ansible Automation Platform needs.

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. INTRODUCTION TO AUTOMATION EXECUTION ENVIRONMENTS

Using Ansible content that depends on non-default dependencies can be complicated because the packages must be installed on each node, interact with other software installed on the host system, and be kept in sync.

Automation execution environments help simplify this process and can easily be created with Ansible Builder.

1.1. ABOUT AUTOMATION EXECUTION ENVIRONMENTS

All automation in Red Hat Ansible Automation Platform runs on container images called automation execution environments. Automation execution environments create a common language for communicating automation dependencies, and offer a standard way to build and distribute the automation environment.

An automation execution environment should contain the following:

- Ansible Core 2.15 or later
- Python 3.8-3.11
- Ansible Runner
- Ansible content collections and their dependencies
- System dependencies

1.1.1. Why use automation execution environments?

With automation execution environments, Red Hat Ansible Automation Platform has transitioned to a distributed architecture by separating the control plane from the execution plane. Keeping automation execution independent of the control plane results in faster development cycles and improves scalability, reliability, and portability across environments. Red Hat Ansible Automation Platform also includes access to Ansible content tools, making it easy to build and manage automation execution environments.

In addition to speed, portability, and flexibility, automation execution environments provide the following benefits:

- They ensure that automation runs consistently across multiple platforms and make it possible to incorporate system-level dependencies and collection-based content.
- They give Red Hat Ansible Automation Platform administrators the ability to provide and manage automation environments to meet the needs of different teams.
- They allow automation to be easily scaled and shared between teams by providing a standard way of building and distributing the automation environment.
- They enable automation teams to define, build, and update their automation environments themselves.
- Automation execution environments provide a common language to communicate automation dependencies.

CHAPTER 2. USING ANSIBLE BUILDER

Ansible Builder is a command line tool that automates the process of building automation execution environments by using metadata defined in various Ansible Collections or created by the user.

2.1. WHY USE ANSIBLE BUILDER?

Before Ansible Builder was developed, Red Hat Ansible Automation Platform users could run into dependency issues and errors when creating custom virtual environments or containers that included all of the required dependencies installed.

Now, with Ansible Builder, you can easily create a customizable automation execution environments definition file that specifies the content you want included in your automation execution environments such as Ansible Core, Python, Collections, third-party Python requirements, and system level packages. This allows you to fulfill all of the necessary requirements and dependencies to get jobs running.

2.2. INSTALLING ANSIBLE BUILDER

Prerequisites

- You have installed the Podman container runtime.
- You have valid subscriptions attached on the host. Doing so allows you to access the subscription-only resources needed to install **ansible-builder**, and ensures that the necessary repository for **ansible-builder** is automatically enabled. See [Attaching your Red Hat Ansible Automation Platform subscription](#) for more information.

Procedure

- In your terminal, run the following command to install Ansible Builder and activate your Ansible Automation Platform repo:

```
# dnf install --enablerepo=ansible-automation-platform-2.5-for-rhel-9-x86_64-rpms ansible-builder
```

2.3. 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. For information about importing collections from Ansible Galaxy or automation hub into a private automation hub, see [Importing an automation content collection in automation hub](#).

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.

2.4. BUILDING A DEFINITION FILE

After you install Ansible Builder, you can create a definition file that Ansible Builder uses to create your automation execution environment image. Ansible Builder makes an automation execution environment image by reading and validating your definition file, then creating a **Containerfile**, and finally passing the **Containerfile** to Podman, which then packages and creates your automation execution environment image. The definition file that you create must be in **yaml** format and contain different sections. The default definition filename, if not provided, is **execution-environment.yml**. For more information on the parts of a definition file, see [Breakdown of definition file content](#).

The following is an example of a version 3 definition file. Each definition file must specify the major version number of the Ansible Builder feature set it uses. If not specified, Ansible Builder defaults to version 1, making most new features and definition keywords unavailable.

Example 2.1. Definition file example

```
version: 3

build_arg_defaults: ❶
  ANSIBLE_GALAXY_CLI_COLLECTION_OPTS: '--pre'

dependencies: ❷
  galaxy: requirements.yml
  python:
    - six
    - psutil
  system: bindep.txt

images: ❸
  base_image:
    name: registry.redhat.io/ansible-automation-platform-24/ee-minimal-rhel9:latest

# Custom package manager path for the RHEL based images
options: ❹
  package_manager_path: /usr/bin/microdnf

additional_build_steps: ❺
```

```

prepend_base:
  - RUN echo This is a prepend base command!

prepend_galaxy:
  # Environment variables used for Galaxy client configurations
  - ENV ANSIBLE_GALAXY_SERVER_LIST=automation_hub
  - ENV
ANSIBLE_GALAXY_SERVER_AUTOMATION_HUB_URL=https://console.redhat.com/api/automati
on-hub/content/xxxxxxx-synclist/
  - ENV
ANSIBLE_GALAXY_SERVER_AUTOMATION_HUB_AUTH_URL=https://sso.redhat.com/auth/real
ms/redhat-external/protocol/openid-connect/token
  # define a custom build arg env passthru - we still also have to pass
  # `--build-arg ANSIBLE_GALAXY_SERVER_AUTOMATION_HUB_TOKEN` to get it to pick it
up from the env
  - ARG ANSIBLE_GALAXY_SERVER_AUTOMATION_HUB_TOKEN

prepend_final: |
  RUN whoami
  RUN cat /etc/os-release
append_final:
  - RUN echo This is a post-install command!
  - RUN ls -la /etc

```

- 1 Lists default values for build arguments.
- 2 Specifies the location of various requirements files.
- 3 Specifies the base image to be used. Red Hat support is only provided for the redhat.registry.io base image.
- 4 Specifies options that can affect builder runtime functionality.
- 5 Commands for additional custom build steps.

Additional resources

- For more information about the definition file content, see [Breakdown of definition file content](#).
- To read more about the differences between Ansible Builder versions 2 and 3, see the [Ansible 3 Porting Guide](#).

2.5. BUILDING THE AUTOMATION EXECUTION ENVIRONMENT IMAGE

After you create a definition file, you can proceed to build an automation execution environment image.



NOTE

When building an execution environment image, it must support the architecture that Ansible Automation Platform is deployed with.

Prerequisites

- You have created a definition file.

Procedure

To build an automation execution environment image, run the following from the command line:

```
$ ansible-builder build
```

By default, Ansible Builder looks for a definition file named **execution-environment.yml** but a different file path can be specified as an argument with the **-f** flag:

```
$ ansible-builder build -f definition-file-name.yml
```

where *definition-file-name* specifies the name of your definition file.

2.6. BREAKDOWN OF DEFINITION FILE CONTENT

A definition file is required for building automation execution environments with Ansible Builder, because it specifies the content that is included in the automation execution environment container image.

The following sections breaks down the different parts of a definition file.

2.6.1. Build args and base image

The **build_arg_defaults** section of the definition file is a dictionary whose keys can provide default values for arguments to Ansible Builder. See the following table for a list of values that can be used in **build_arg_defaults**:

Value	Description
ANSIBLE_GALAXY_CLI_COLLECTION_OPTS	Allows the user to pass arbitrary arguments to the ansible-galaxy CLI during the collection installation phase. For example, the <code>-pre</code> flag to enable the installation of pre-release collections, or <code>-c</code> to disable verification of the server's SSL certificate.
ANSIBLE_GALAXY_CLI_ROLE_OPTS	Allows the user to pass any flags, such as <code>-no-deps</code> , to the role installation.

The values given inside **build_arg_defaults** will be hard-coded into the **Containerfile**, so these values will persist if **podman build** is called manually.



NOTE

If the same variable is specified in the CLI **--build-arg** flag, the CLI value will take higher precedence.

You can include dependencies that must be installed into the final image in the dependencies section of your definition file.

To avoid issues with your automation execution environment image, make sure that the entries for Galaxy, Python, and system point to a valid requirements file, or are valid content for their respective file types.

2.6.1.1. Galaxy

The **galaxy** entry points to a valid requirements file or includes inline content for the **ansible-galaxy collection install -r ...** command.

The entry **requirements.yml** can be a relative path from the directory of the automation execution environment definition's folder, or an absolute path.

The content might look like the following:

Example 2.2. Galaxy entry

```
collections:
- community.aws
- kubernetes.core
```

2.6.1.2. Python

The **python** entry in the definition file points to a valid requirements file or to an inline list of Python requirements in PEP508 format for the **pip install -r ...** command.

The entry **requirements.txt** is a file that installs extra Python requirements on top of what the Collections already list as their Python dependencies. It may be listed as a relative path from the directory of the automation execution environment definition's folder, or an absolute path. The contents of a **requirements.txt** file should be formatted like the following example, similar to the standard output from a **pip freeze** command:

Example 2.3. Python entry

```
boto>=2.49.0
botocore>=1.12.249
pytz
python-dateutil>=2.7.0
awxkit
packaging
requests>=2.4.2
xmltodict
azure-cli-core==2.11.1
openshift>=0.6.2
requests-oauthlib
openstacksdk>=0.13
ovirt-engine-sdk-python>=4.4.10
```

2.6.1.3. System

The **system** entry in the definition points to a [bindep](#) requirements file or to an inline list of bindep entries, which install system-level dependencies that are outside of what the collections already include

as their dependencies. It can be listed as a relative path from the directory of the automation execution environment definition's folder, or as an absolute path. At a minimum, the the collection(s) must specify necessary requirements for **[platform:rpm]**.

To demonstrate this, the following is an example **bindep.txt** file that adds the **libxml2** and **subversion** packages to a container:

Example 2.4. System entry

```
libxml2-devel [platform:rpm]
subversion [platform:rpm]
```

Entries from multiple collections are combined into a single file. This is processed by **bindep** and then passed to **dnf**. Only requirements with no profiles or no runtime requirements will be installed to the image.

2.6.2. Images

The **images** section of the definition file identifies the base image. Verification of signed container images is supported with the **podman** container runtime.

See the following table for a list of values that you can use in **images**:

Value	Description
base_image	<p>Specifies the parent image for the automation execution environment which enables a new image to be built that is based on an existing image. This is typically a supported execution environment base image such as <i>ee-minimal</i> or <i>ee-supported</i>, but it can also be an execution environment image that you have created and want to customize further.</p> <p>A name key is required for the container image to use. Specify the signature_original_name key if the image is mirrored within your repository, but is signed with the image's original signature key. Image names must contain a tag, such as :latest.</p> <p>The default image is registry.redhat.io/ansible-automation-platform-24/ee-minimal-rhel8:latest.</p>



NOTE

If the same variable is specified in the CLI **--build-arg** flag, the CLI value will take higher precedence.

2.6.3. Additional build files

You can add any external file to the build context directory by referring or copying them to the **additional_build_steps** section of the definition file. The format is a list of dictionary values, each with a **src** and **dest** key and value.

Each list item must be a dictionary containing the following required keys:

src

Specifies the source files to copy into the build context directory. This can be an absolute path (for example, **/home/user/.ansible.cfg**), or a path that is relative to the execution environment file. Relative paths can be glob expressions matching one or more files (for example, **files/*.cfg**).



NOTE

Absolute paths can not include a regular expression. If **src** is a directory, the entire contents of that directory are copied to **dest**.

dest

Specifies a subdirectory path underneath the **_build** subdirectory of the build context directory that contains the source files (for example, **files/configs**). This can not be an absolute path or contain **..** within the path. Ansible Builder creates this directory for you if it does not already exist.

2.6.4. Additional custom build steps

You can specify custom build commands for any build phase in the **additional_build_steps** section of the definition file. This allows fine-grained control over the build phases.

Use the **prepend_** and **append_** commands to add directives to the **Containerfile** that run either before or after the main build steps are executed. The commands must conform to any rules required for the runtime system.

See the following table for a list of values that can be used in **additional_build_steps**:

Value	Description
prepend_base	Allows you to insert commands before building the base image.
append_base	Allows you to insert commands after building the base image.
prepend_galaxy	Allows you to insert before building the galaxy image.
append_galaxy	Allows you to insert after building the galaxy image.
prepend_builder	Allows you to insert commands before building the Python builder image.
append_builder	Allows you to insert commands after building the Python builder image.
prepend_final	Allows you to insert before building the final image.

Value	Description
append_final	Allows you to insert after building the final image.

The syntax for **additional_build_steps** supports both multi-line strings and lists. See the following examples:

Example 2.5. A multi-line string entry

```
prepend_final: |
  RUN whoami
  RUN cat /etc/os-release
```

Example 2.6. A list entry

```
append_final:
  - RUN echo This is a post-install command!
  - RUN ls -la /etc
```

2.6.5. Additional resources

- For example definition files for common scenarios, see the [Common scenarios section](#) of the *Ansible Builder Documentation*

2.7. OPTIONAL BUILD COMMAND ARGUMENTS

The **-t** flag will tag your automation execution environment image with a specific name. For example, the following command will build an image named **my_first_ee_image**:

```
$ ansible-builder build -t my_first_ee_image
```



NOTE

If you do not use **-t** with **build**, an image called **ansible-execution-env** is created and loaded into the local container registry.

If you have multiple definition files, you can specify which one to use by including the **-f** flag:

```
$ ansible-builder build -f another-definition-file.yml -t another_ee_image
```

In [this example](#), Ansible Builder will use the specifications provided in the file named **another-definition-file.yml** instead of the default **execution-environment.yml** to build an automation execution environment image named **another_ee_image**.

For other specifications and flags that you can use with the **build** command, enter **ansible-builder build --help** to see a list of additional options.

2.8. CONTAINERFILE

After your definition file is created, Ansible Builder reads and validates it, creates a **Containerfile** and container build context, and optionally passes these to Podman to build your automation execution environment image. The container build occurs in several distinct stages: **base**, **galaxy**, **builder**, and **final**. The image build steps (along with any corresponding custom **prepend_** and **append_** steps defined in **additional_build_steps**) are:

1. During the **base** build stage, the specified base image is (optionally) customized with components required by other build stages, including Python, **pip**, **ansible-core**, and **ansible-runner**. The resulting image is then validated to ensure that the required components are available (as they may have already been present in the base image). Ephemeral copies of the resulting customized **base** image are used as the base for all other build stages.
2. During the **galaxy** build stage, collections specified by the definition file are downloaded and stored for later installation during the **final** build stage. Python and system dependencies declared by the collections, if any, are also collected for later analysis.
3. During the **builder** build stage, Python dependencies declared by collections are merged with those listed in the definition file. This final set of Python dependencies is downloaded and built as Python wheels and stored for later installation during the **final** build stage.
4. During the **final** build stage, the previously-downloaded collections are installed, along with system packages and any previously-built Python packages that were declared as dependencies by the collections or listed in the definition file.

2.9. CREATING A CONTAINERFILE WITHOUT BUILDING AN IMAGE

If you are required to use shared container images built in sandboxed environments for security reasons, you can create a shareable **Containerfile**.

```
$ ansible-builder create
```

CHAPTER 3. COMMON AUTOMATION EXECUTION ENVIRONMENT SCENARIOS

Use the following example definition files to address common configuration scenarios.

3.1. UPDATING THE AUTOMATION HUB CA CERTIFICATE

Use this example to customize the default definition file to include a CA certificate to the **additional-build-files** section, move the file to the appropriate directory and, finally, run the command to update the dynamic configuration of CA certificates to allow the system to trust this CA certificate.

Prerequisites

- A custom CA certificate, for example **rootCA.crt**.



NOTE

Customizing the CA certificate using **prepend_base** means that the resulting CA configuration appears in all other build stages and the final image, because all other build stages inherit from the base image.

```
additional_build_files:
  # copy the CA public key into the build context, we will copy and use it in the base image later
  - src: files/rootCA.crt
    dest: configs

additional_build_steps:
  prepend_base:
    # copy a custom CA cert into the base image and recompute the trust database
    # because this is in "base", all stages will inherit (including the final EE)
    - COPY _build/configs/rootCA.crt /usr/share/pki/ca-trust-source/anchors
    - RUN update-ca-trust

options:
  package_manager_path: /usr/bin/microdnf # downstream images use non-standard package
  manager

[galaxy]
server_list = automation_hub
```

3.2. USING AUTOMATION HUB AUTHENTICATION DETAILS WHEN BUILDING AUTOMATION EXECUTION ENVIRONMENTS

Use the following example to customize the default definition file to pass automation hub authentication details into the automation execution environment build without exposing them in the final automation execution environment image.

Prerequisites

- You have created an API token, as in [Retrieving the API token for your Red Hat Certified Collection](#) and stored it in a secure location, for example in a file named **token.txt**.

- Define a build argument that gets populated with the automation hub API token:

```
export ANSIBLE_GALAXY_SERVER_AUTOMATION_HUB_TOKEN=$(cat <token.txt>)
```

```
additional_build_steps:
```

```
  prepend_galaxy:
```

```
    # define a custom build arg env passthru- we still also have to pass
```

```
    # `--build-arg ANSIBLE_GALAXY_SERVER_AUTOMATION_HUB_TOKEN` to get it to pick it up  
from the host env
```

```
  - ARG ANSIBLE_GALAXY_SERVER_AUTOMATION_HUB_TOKEN
```

```
  - ENV ANSIBLE_GALAXY_SERVER_LIST=automation_hub
```

```
  - ENV
```

```
ANSIBLE_GALAXY_SERVER_AUTOMATION_HUB_URL=https://console.redhat.com/api/automation-  
hub/content/<yourhuburl>-synclist/
```

```
  - ENV
```

```
ANSIBLE_GALAXY_SERVER_AUTOMATION_HUB_AUTH_URL=https://sso.redhat.com/auth/realms/  
redhat-external/protocol/openid-connect/token
```

3.3. ADDITIONAL RESOURCES

- For information regarding the different parts of an automation execution environment definition file, see [Breakdown of definition file content](#).
- For additional example definition files for common scenarios, see [Common scenarios section](#) of the *Ansible Builder Documentation*

CHAPTER 4. PUBLISHING AN AUTOMATION EXECUTION ENVIRONMENT

4.1. CUSTOMIZING AN EXISTING AUTOMATION EXECUTION ENVIRONMENTS IMAGE

Ansible Controller includes the following default execution environments:

- **Minimal** - Includes the latest Ansible-core 2.15 release along with Ansible Runner, but does not include collections or other content
- **EE Supported** - Minimal, plus all Red Hat-supported collections and dependencies

While these environments cover many automation use cases, you can add additional items to customize these containers for your specific needs. The following procedure adds the **kubernetes.core** collection to the **ee-minimal** default image:

Procedure

1. Log in to **registry.redhat.io** via Podman:

```
$ podman login -u="[username]" -p="[token/hash]" registry.redhat.io
```

2. Ensure that you can pull the required automation execution environment base image:

```
podman pull registry.redhat.io/ansible-automation-platform-24/ee-minimal-rhel8:latest
```

3. Configure your Ansible Builder files to specify the required base image and any additional content to add to the new execution environment image.

- a. For example, to add the [Kubernetes Core Collection from Galaxy](#) to the image, use the Galaxy entry:

```
collections:  
- kubernetes.core
```

- b. For more information about definition files and their content, see the [definition file breakdown section](#).

4. In the execution environment definition file, specify the original **ee-minimal** container's URL and tag in the **EE_BASE_IMAGE** field. In doing so, your final **execution-environment.yml** file will look like the following:

Example 4.1. A customized **execution-environment.yml** file

```
version: 3  
  
images:  
  base_image: 'registry.redhat.io/ansible-automation-platform-25/ee-minimal-rhel9:latest'  
  
dependencies:
```

```
galaxy:
collections:
- kubernetes.core
```

**NOTE**

Since this example uses the community version of **kubernetes.core** and not a certified collection from automation hub, we do not need to create an **ansible.cfg** file or reference that in our definition file.

- Build the new execution environment image by using the following command:

```
$ ansible-builder build -t [username]/new-ee
```

where **[username]** specifies your username, and **new-ee** specifies the name of your new container image.

**NOTE**

If you do not use **-t** with **build**, an image called **ansible-execution-env** is created and loaded into the local container registry.

- Use the **podman images** command to confirm that your new container image is in that list:

Example 4.2. Output of a podman images command with the imagenew-ee

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
localhost/new-ee	latest	f5509587efbb	3 minutes ago	769 MB

- Verify that the collection is installed:

```
$ podman run [username]/new-ee ansible-doc -l kubernetes.core
```

- Tag the image for use in your automation hub:

```
$ podman tag [username]/new-ee [automation-hub-IP-address]/[username]/new-ee
```

- Log in to your automation hub using Podman:

**NOTE**

You must have **admin** or appropriate container repository permissions for automation hub to push a container. For more information, see [Manage containers in private automation hub](#).

```
$ podman login -u="[username]" -p="[token/hash]" [automation-hub-IP-address]
```

- Push your image to the container registry in automation hub:

```
$ podman push [automation-hub-IP-address]/[username]/new-ee
```

10. Pull your new image into your automation controller instance:
 - a. Go to automation controller.
 - b. From the navigation panel, select **Automation Execution** → **Infrastructure** → **Execution Environments**.
 - c. Click **Add**.
 - d. Enter the appropriate information then click **Save** to pull in the new image.



NOTE

If your instance of automation hub is password or token protected, ensure that you have the appropriate container registry credential set up.

4.2. ADDITIONAL RESOURCES (OR NEXT STEPS)

For more details on customizing execution environments based on common scenarios, see the following topics in the *Ansible Builder Documentation*:

- [Copying arbitrary files to an execution environment](#)
- [Building execution environments with environment variables](#)
- [Building execution environments with environment variables and **ansible.cfg**](#)

CHAPTER 5. POPULATING YOUR PRIVATE AUTOMATION HUB CONTAINER REGISTRY

By default, private automation hub does not include execution environments. To populate your container registry, you must push a execution environment to it.

You must follow a specific workflow to populate your private automation hub remote registry:

- Pull execution environments from the Red Hat Ecosystem Catalog (registry.redhat.io)
- Tag them
- Push them to your private automation hub remote registry

IMPORTANT

Image manifests and filesystem blobs were both originally served directly from **registry.redhat.io** and **registry.access.redhat.com**. As of 1 May 2023, filesystem blobs are served from **quay.io** instead.

- Ensure that the [Network ports and protocols](#) listed in *Table 5.4. Execution Environments (EE)* are available to avoid problems pulling container images.

Make this change to any firewall configuration that specifically enables outbound connections to **registry.redhat.io** or **registry.access.redhat.com**.

Use the hostnames instead of IP addresses when configuring firewall rules.

After making this change you can continue to pull execution environments from **registry.redhat.io** and **registry.access.redhat.com**. You do not require a **quay.io** login, or need to interact with the **quay.io** registry directly in any way to continue pulling Red Hat container images.

However, manifests, sometimes called “subscription allocations”, on the web-based Red Hat Subscription Management are no longer supported as of early 2024 with one exception: If a system is part of a closed network or “air gapped” system that does not receive its updates from Red Hat’s servers directly, manifests are supported until the release of Red Hat Satellite 6.16. Keep up to date with [Red Hat Satellite Release Dates](#) for the announcement for Red Hat Satellite 6.16’s release date announcement.

5.1. PULLING EXECUTION ENVIRONMENTS FOR USE IN AUTOMATION HUB

Before you can push execution environments to your private automation hub, you must first pull them from an existing registry and tag them for use. The following example details how to pull an execution environment from the Red Hat Ecosystem Catalog (registry.redhat.io).



IMPORTANT

As of early 2024, Red Hat no longer supports manifests or manifest lists on the Red Hat Subscription Management web platform, which has also been used interchangeably with “subscription allocations.” Red Hat also no longer supports most manifest functionality in Red Hat Satellite with one exception: * Red Hat Satellite users in closed network or “air gapped” networks that do not receive their updates directly from Red Hat servers can currently still use **access.redhat.com** until the release of Red Hat Satellite 6.16.

New Red Hat accounts automatically use Simple Content Access for their subscription tooling. New Red Hat accounts and existing Satellite customers who can connect to Red Hat’s servers can find their manifests at **console.redhat.com**.

Prerequisites

- You have permissions to pull execution environments from registry.redhat.io.
- A Red Hat account with Simple Content Access enabled.

Procedure

1. If you need to access your manifest for your container images log in to [Red Hat Console](#).
2. Click the three-dot menu for the manifest you need for your execution environments, and click **Export manifest**.
3. Log in to Podman by using your registry.redhat.io credentials:

```
$ podman login registry.redhat.io
```

4. Enter your username and password.
5. Pull an execution environment:

```
$ podman pull registry.redhat.io/<ee_name>:<tag>
```

Verification

To verify that the execution environment you recently pulled is contained in the list, take these steps:

1. List the images in local storage:

```
$ podman images
```

2. Check the execution environment name, and verify that the tag is correct.

Additional resources

- See [Red Hat Ecosystem Catalog Help](#) for information on registering and getting execution environments.
- See [Creating and managing manifests for a connected Satellite Server](#) to learn more about the changes coming to Red Hat subscription tooling.

5.2. TAGGING EXECUTION ENVIRONMENTS FOR USE IN AUTOMATION HUB

After you pull execution environments from a registry, tag them for use in your private automation hub remote registry.

Prerequisites

- You have pulled an execution environment from an external registry.
- You have the FQDN or IP address of the automation hub instance.

Procedure

- Tag a local execution environment with the automation hub container repository:

```
$ podman tag registry.redhat.io/<ee_name>:<tag>  
<automation_hub_hostname>/<ee_name>
```

Verification

1. List the images in local storage:

```
$ podman images
```

2. Verify that the execution environment you recently tagged with your automation hub information is contained in the list.

5.3. PUSHING AN EXECUTION ENVIRONMENT TO PRIVATE AUTOMATION HUB

You can push tagged execution environments to private automation hub to create new containers and populate the remote registry.

Prerequisites

- You have permissions to create new containers.
- You have the FQDN or IP address of the automation hub instance.

Procedure

1. Log in to Podman using your automation hub location and credentials:

```
$ podman login -u=<username> -p=<password> <automation_hub_url>
```



WARNING

Let Podman prompt you for your password when you log in. Entering your password at the same time as your username can expose your password to the shell history.

2. Push your execution environment to your automation hub remote registry:

```
$ podman push <automation_hub_url>/<ee_name>
```

Troubleshooting

The **push** operation re-compresses image layers during the upload, which is not guaranteed to be reproducible and is client-implementation dependent. This may lead to image-layer digest changes and a failed push operation, resulting in **Error: Copying this image requires changing layer representation, which is not possible (image is signed or the destination specifies a digest)**.

Verification

1. Log in to your Ansible Automation Platform.
2. Navigate to **Automation Content** → **Execution Environments**.
3. Locate the container in the container repository list.

CHAPTER 6. SETTING UP YOUR CONTAINER REPOSITORY

When you set up your container repository, you must add a description, include a README, add teams that can access the repository, and tag automation execution environments.

6.1. PREREQUISITES TO SETTING UP YOUR REMOTE REGISTRY

- You are logged in to Ansible Automation Platform.
- You have permissions to change the repository.

6.2. ADDING A README TO YOUR CONTAINER REPOSITORY

Add a README to your container repository to provide instructions to your users on how to work with the container. Automation hub container repositories support Markdown for creating a README. By default, the README is empty.

Prerequisites

- You have permissions to change containers.

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Automation Content** → **Execution Environments**.
3. Select your execution environment.
4. On the **Detail** tab, click **Add**.
5. In the **Raw Markdown** text field, enter your README text in Markdown.
6. Click **Save** when you are finished.

After you add a README, you can edit it at any time by clicking **Edit** and repeating steps 4 and 5.

6.3. PROVIDING ACCESS TO YOUR AUTOMATION EXECUTION ENVIRONMENTSS

Provide access to your automation execution environments for users who need to work with the images. Adding a team allows you to modify the permissions the team can have to the container repository. You can use this option to extend or restrict permissions based on what the team is assigned.

Prerequisites

- You have **change container namespace** permissions.

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Automation Content** → **Execution Environments**.

3. Select your automation execution environments.
4. From the **Team Access** tab, click **Add roles**.
5. Select the team or teams to which you want to grant access and click **Next**.
6. Select the roles that you want to add to this execution environment and click **Next**.
7. Click **Finish**.

6.4. TAGGING CONTAINER IMAGES

Tag automation execution environments to add an additional name to automation execution environments stored in your automation hub container repository. If no tag is added to an automation execution environment, automation hub defaults to **latest** for the name.

Prerequisites

- You have **change automation execution environments tags** permissions.

Procedure

1. From the navigation panel, select **Automation Content** → **Execution Environments**.
2. Select your automation execution environments.
3. Click the **Images** tab.
4. Click the **More Actions** icon **:**, and click **Manage tags**.
5. Add a new tag in the text field and click **Add**.
6. Optional: Remove **current tags** by clicking **x** on any of the tags for that image.

Verification

- Click the **Activity** tab and review the latest changes.

6.5. CREATING A CREDENTIAL

To pull automation execution environment images from a password or token-protected registry, you must create a credential.

In earlier versions of Ansible Automation Platform, you were required to deploy a registry to store execution environment images. On Ansible Automation Platform 2.0 and later, the system operates as if you already have a remote registry up and running. To store execution environment images, add the credentials of only your selected remote registries.

Procedure

1. Log in to Ansible Automation Platform.
2. From the navigation panel, select **Automation Execution** → **Infrastructure** → **Credentials**.
3. Click **Create credential** to create a new credential.

4. Enter an authorization **Name**, **Description**, and **Organization**.
5. In the **Credential Type** drop-down, select **Container Registry**.
6. Enter the **Authentication URL**. This is the remote registry address.
7. Enter the **Username** and **Password or Token** required to log in to the remote registry.
8. Optional: To enable SSL verification, select **Verify SSL**.
9. Click **Create credential**.

Filling in at least one of the fields organization, user, or team is mandatory, and can be done through the user interface.

CHAPTER 7. PULLING IMAGES FROM A CONTAINER REPOSITORY

Pull automation execution environments from the automation hub remote registry to make a copy to your local machine. Automation hub provides the **podman pull** command for each **latest** automation execution environments in the container repository. You can copy and paste this command into your terminal, or use **podman pull** to copy an automation execution environments based on an automation execution environments tag.

7.1. PULLING AN IMAGE

You can pull automation execution environments from the automation hub remote registry to make a copy to your local machine.

Prerequisites

- You must have permission to view and pull from a private container repository.

Procedure

1. If you are pulling automation execution environments from a password or token-protected registry, [create a credential](#) before pulling the automation execution environments.
2. From the navigation panel, select **Automation Content** → **Execution Environments**.
3. Select your automation execution environments.
4. In the **Pull this image** entry, click **Copy to clipboard**.
5. Paste and run the command in your terminal.

Verification

- Run **podman images** to view images on your local machine.

7.2. SYNCING IMAGES FROM A CONTAINER REPOSITORY

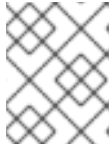
You can pull automation execution environments from the automation hub remote registry to sync an image to your local machine. To sync an automation execution environments from a remote registry, you must first configure a remote registry.

Prerequisites

You must have permission to view and pull from a private container repository.

Procedure

1. From the navigation panel, select **Automation Content** → **Execution Environments**.
2. Add <https://registry.redhat.io> to the registry.
3. Add any required credentials to authenticate.

**NOTE**

Some remote registries are aggressive with rate limiting. Set a rate limit under **Advanced Options**.

4. From the navigation panel, select **Automation Content** → **Execution Environments**.
5. Click **Create execution environment** in the page header.
6. Select the registry you want to pull from. The **Name** field displays the name of the automation execution environments displayed on your local registry.

**NOTE**

The **Upstream name** field is the name of the image on the remote server. For example, if the upstream name is set to "alpine" and the **Name** field is "local/alpine", the alpine image is downloaded from the remote and renamed to "local/alpine".

7. Set a list of tags to include or exclude. Syncing automation execution environments with a large number of tags is time consuming and uses a lot of disk space.

Additional resources

- See [Red Hat Container Registry Authentication](#) for a list of registries.
- See the [What is Podman?](#) documentation for options to use when pulling images.

PART I. OPEN SOURCE LICENSE

Apache License

Version 2.0, January 2004

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

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

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

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

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

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

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

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

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

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

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

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

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor

hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

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

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

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

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

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

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

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT

WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

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

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

END OF TERMS AND CONDITIONS

APPENDIX A. AUTOMATION EXECUTION ENVIRONMENTS PRECEDENCE

Project updates will always use the control plane automation execution environments by default, however, jobs will use the first available automation execution environments as follows:

1. The **execution_environment** defined on the template (job template or inventory source) that created the job.
2. The **default_environment** defined on the project that the job uses.
3. The **default_environment** defined on the organization of the job.
4. The **default_environment** defined on the organization of the inventory the job uses.
5. The current **DEFAULT_EXECUTION_ENVIRONMENT** setting (configurable at **api/v2/settings/system/**)
6. Any image from the **GLOBAL_JOB_EXECUTION_ENVIRONMENTS** setting.
7. Any other global execution environment.



NOTE

If more than one execution environment fits a criteria (applies for 6 and 7), the most recently created one is used.