Red Hat Ansible Automation Platform 2.2

Creating and Consuming Execution Environments

Create consistent and reproducible automation execution environments for your Red Hat Ansible Automation Platform.

Red Hat Ansible Automation Platform 2.2 Creating and Consuming Execution Environments

Create consistent and reproducible automation execution environments for your Red Hat Ansible Automation Platform.

Legal Notice

Abstract

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

Table of Contents

PREFACE

Use Ansible Builder to create consistent and reproducible automation execution environments for your Red Hat Ansible Automation Platform needs.

MAKING OPEN SOURCE MORE INCLUSIVE

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

CHAPTER 1. 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

Automation execution environments are container images on which all automation in Red Hat Ansible Automation Platform is run. Automation execution environments create a common language for communicating automation dependencies, and provide a standard way to build and distribute the automation environment.

An automation execution environment is expected to contain the following:

- Ansible 2.9 or Ansible Core 2.11-2.13

- Python 3.8-3.10

- Ansible Runner

- Ansible content collections

- Collection, Python, or 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, collections, 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

You can install Ansible Builder using Red Hat Subscription Management (RHSM) to attach your Red Hat Ansible Automation Platform subscription. Attaching your Red Hat Ansible Automation Platform subscription allows you to access subscription–only resources necessary to install **ansible-builder**. Once you attach your subscription, the necessary repository for **ansible-builder** is automatically enabled.

> **NOTE**
>
> You must have valid subscriptions attached on the host before installing **ansible-builder**.

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-2.2-for-rhel-8-x86_64-rpms ansible-builder
  ```

2.3. BUILDING A DEFINITION FILE

Once you have Ansible Builder installed, you can create a definition file that Ansible Builder uses to create your automation execution environment image. The high level process to build an automation execution environment image is for Ansible Builder to read and validate your definition file, then create a **Containerfile**, and finally pass the **Containerfile** to Podman which then packages and creates your automation execution environment image. The definition file created is in **yaml** format and contains different sections. For more information about the definition file content, see Breakdown of definition file content.

The following is an example of a definition file:

Example 2.1. A definition file

```
version: 1

build_arg_defaults: ❶
  ANSIBLE_GALAXY_CLI_COLLECTION_OPTS: "-v"
```

```
dependencies: ❷
  galaxy: requirements.yml
  python: requirements.txt
  system: bindep.txt

additional_build_steps: ❸
  prepend: |
    RUN whoami
    RUN cat /etc/os-release
  append:
    - RUN echo This is a post-install command!
    - RUN ls -la /etc
```

❶ Lists default values for build arguments

❷ Specifies the location of various requirements files

❸ Commands for additional custom build steps

For more information about these definition file parameters, see Breakdown of definition file content .

2.4. EXECUTING THE BUILD AND CREATING COMMANDS

Prerequisites

- You have created a definition file

Procedure

To build an automation execution environment image, run:

```
$ ansible-builder build
```

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

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

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

2.5. 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.

2.5.1. 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 –pre flag to enable the installation of pre-release collections, or -c to disable verification of the server's SSL certificate.
EE_BASE_IMAGE	Specifies the parent image for the automation execution environment, enabling a new image to be built that is based off of an already-existing image. This is typically a supported execution environment base image such as ee-minimal or ee-supported, but it can also be an execution environment image that you have created previously and want to customize further. The default image is **registry.redhat.io/ansible-automation-platform-23/ee-minimal-rhel8:latest**.
EE_BUILDER_IMAGE	Specifies the intermediate builder image used for Python dependency collection and compilation; must contain a matching Python version with **EE_BASE_IMAGE** and have ansible-builder installed. The default image is **registry.redhat.io/ansible-automation-platform-23/ansible-builder-rhel8:latest**.

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.

2.5.2. Ansible config file path

The **ansible_config** directive allows specifying the path to an **ansible.cfg** file to pass a token and other settings for a private account to an automation hub server during the Collection installation stage of the build. The config file path should be relative to the definition file location, and will be copied to the generated container build context.

The **ansible.cfg** file should be formatted like the following example:

Example 2.2. An **ansible.cfg** file

```
[galaxy]
server_list = automation_hub

[galaxy_server.automation_hub]
url=https://cloud.redhat.com/api/automation-hub/
auth_url=https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token
token=my_ah_token
```

For more information on how to download a collection from automation hub, please see the related Ansible documentation page.

2.5.3. Dependencies

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.

2.5.3.1. Galaxy

The **galaxy** entry points to a valid requirements file for the **ansible-galaxy collection install -r …** command.

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

The content of a **requirements.yml** file may look like the following:

Example 2.3. A **requirements.yml** file for Galaxy

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

2.5.3.2. Python

The **python** entry in the definition file points to a valid requirements file 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 2.4. A **requirements.txt** file for Python

```
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
python_version >= '2.7'
collection community.vmware
google-auth
openshift>=0.6.2
requests-oauthlib
openstacksdk>=0.13
ovirt-engine-sdk-python>=4.4.10
```

2.5.3.3. System

The **system** entry in the definition points to a bindep requirements file, which will 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 an absolute path. A minimum expectation is that the collection(s) 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 2.5. A **bindep.txt** file

```
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.

2.5.4. Additional custom build steps

The **prepend** and **append** commands may be specified in the **additional_build_steps section**. These will add commands to the **Containerfile** which will run either before or after the main build steps are executed.

The syntax for **additional_build_steps** must be one of the following:

- a multi-line string

  Example 2.6. A multi-line string entry

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

- a list

  Example 2.7. A list entry

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

2.6. 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 utilizing the **-f** flag:

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

In the example above, Ansible Builder will use the specifications provided in the file **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 are possible to use with the build command, enter **ansible-builder build --help** to see a list of additional options.

2.7. CONTAINERFILE

Once your definition file is created, Ansible Builder reads and validates it, then creates a **Containerfile**, and finally passes the **Containerfile** to Podman to package and create your automation execution environment image using the following instructions:

1. Fetch base image

2. In the ephemeral copy of base image, collections are downloaded and the list of declared Python and system dependencies, if any, are collected for later.

3. In the ephemeral builder image, Python wheels for all Python dependencies listed in the definition file are downloaded and built (as needed), including all Python dependencies declared by collections listed in the definition file.

4. **prepend** for additional_build_steps from the definition file are run.

5. In the final automation execution environments image, system dependencies listed in the definition file are installed, including all system dependencies declared by collections listed in the definition file.

6. In the final automation execution environments image, the downloaded collections are copied and the previously fetched Python dependencies are installed.

7. **append** for additional_build_steps from the definition file are run.

2.8. CREATING A CONTAINERFILE WITHOUT BUILDING AN IMAGE

To create a shareable **Containerfile** without building an image from it, run:

```
$ ansible-builder create
```

CHAPTER 3. PUBLISHING AN AUTOMATION EXECUTION ENVIRONMENT

3.1. CUSTOMIZING AN EXISTING AUTOMATION EXECUTION ENVIRONMENTS IMAGE

Ansible Controller includes three default execution environments:

- **Ansible 2.9** – no collections are installed other than Controller modules

- **Minimal** – contains the latest Ansible 2.13 release along with Ansible Runner, but contains no collections or other additional 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 desired automation execution environment base image

   ```
   podman pull registry.redhat.io/ansible-automation-platform-22/ee-minimal-rhel8:latest
   ```

3. Configure your Ansible Builder files to specify the desired 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, fill out the **requirements.yml** file as follows:

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

   b. For more information on definition files and their content, refer to to 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 3.1. A customized **execution-environment.yml** file

   ```
   version: 1

   build_arg_defaults:
     EE_BASE_IMAGE: 'registry.redhat.io/ansible-automation-platform-22/ee-minimal-rhel8:latest'
   ```

```
dependencies:
  galaxy: requirements.yml
```

> **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.

5. Build the new execution environment image using the following command:

```
$ ansible-builder build -t registry.redhat.io/[username]/new-ee
```

where **[username]** specifies your username, and **new-ee** specifies the name of your new container image.
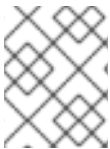
> **NOTE**
>
> If you do not use **-t** with **build**, an image called **ansible-execution-env** is created and loaded into the local container registry.

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

Example 3.2. Output of a podman images command with the image new-ee

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

1. Verify that the collection is installed:

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

2. Tag the image for use in your automation hub:

```
$ podman tag registry.redhat.io/[username]/new-ee [automation-hub-IP-address]/[username]/new-ee
```

3. 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. See *Managing containers in private automation hub* in the Red Hat Ansible Automation Platform documentation for more information.

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

4. Push your image to the container registry in automation hub:

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

5. Pull your new image into your automation controller instance:

b. Navigate to automation controller.

c. From the side-navigational bar, click **Administration → Execution Environments**.

d. Click **Add**.

e. 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.

CHAPTER 4. POPULATING YOUR PRIVATE AUTOMATION HUB CONTAINER REGISTRY

By default, private automation hub does not include container images. To populate your container registry, you need to push a container image to it. The procedures in this section describe how to pull images from the Red Hat Ecosystem Catalog (registry.redhat.io), tag them, and push them to your private automation hub container registry.

4.1. PREREQUISITES

- You have permissions to create new containers and push containers to private automation hub.

4.2. OBTAINING IMAGES FOR USE IN AUTOMATION HUB

Before you can push container images to your private automation hub, you must first pull them from an existing registry and tag them for use. This example details how to pull an image from the Red Hat Ecosystem Catalog (registry.redhat.io).

Prerequisites

You have permissions to pull images from registry.redhat.io.

Procedure

1. Log in to Podman using your registry.redhat.io credentials:

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

2. Enter your username and password at the prompts.

3. Pull a container image:

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

Verification

1. List the images in local storage:

   ```
   $ podman images
   ```

2. Verify that the image you recently pulled is contained in the list.

3. Verify that the tag is correct.

Additional resources

- See Red Hat Ecosystem Catalog Help for information on registering and getting images.

4.3. TAGGING IMAGES FOR USE IN AUTOMATION HUB

After you pull images from a registry, tag them for use in your private automation hub container registry.

Prerequisites

- You have pulled a container image from an external registry.

- You have the FQDN or IP address of the automation hub instance.

Procedure

- Tag a local image with the automation hub container repository

  ```
  $ podman tag registry.redhat.io/<container_image_name>:<tag>
  <automation_hub_URL>/<container_image_name>
  ```

Verification

1. List the images in local storage:

   ```
   $ podman images
   ```

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

4.4. PUSHING A CONTAINER IMAGE TO PRIVATE AUTOMATION HUB

You can push tagged container images to private automation hub to create new containers and populate the container 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>
   ```

2. Push your container image to your automation hub container registry:

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

> **NOTE**
>
> The **--remove-signatures** flag is required when signed images from registry.redhat.io are pushed to the automation hub container registry. 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 automation hub.

2. Navigate to **Container Registry**.

3. Locate the container in the container repository list.

CHAPTER 5. SETTING UP YOUR CONTAINER REPOSITORY

You can setup your container repository to add a description, include a README, add groups who can access the repository, and tag images.

5.1. PREREQUISITES

- You're logged in to a private Automation Hub with permissions to change the repository.

5.2. ADDING A README TO YOUR CONTAINER REPOSITORY

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

Prerequisites

- You have permissions to change containers.

Procedure

1. Navigate to **Execution Environments**.

2. Select your container repository.

3. On the **Detail** tab, click **Add**.

4. In the **Raw Markdown** text field, enter your README text in Markdown.

5. Click **Save** when finished.

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

5.3. PROVIDING ACCESS TO YOUR CONTAINER REPOSITORY

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

Prerequisites

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

Procedure

1. Navigate to **Execution Environments**.

2. Select your container repository.

3. Click **Edit** at the top right of your window.

4. Under **Groups with access**, select a group or groups to grant access to.

- Optional: Add or remove permissions for a specific group using the drop down under that group name.

5. Click **Save**.

5.4. TAGGING CONTAINER IMAGES

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

Prerequisites

- You have **change image tags** permissions.

Procedure

1. Navigate to **Execution Environments**.

2. Select your container repository.

3. Click the **Images** tab.

4. Click   , then click **Manage tags**.

5. Add a new tag in the text field and click **Add**.

   - Optional: Remove **current tags** by clicking the **x** on any of the tags for that image.

6. Click **Save**.

Verification

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

CHAPTER 6. PULLING IMAGES FROM A CONTAINER REPOSITORY

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

6.1. PREREQUISITES

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

6.2. PULLING AN IMAGE

You can pull images from the automation hub container registry to make a copy to your local machine. Automation hub provides the **podman pull** command for each **latest** image in the container repository.

Procedure

1. Navigate to **Execution Environments**.

2. Select your container repository.

3. In the **Pull this image** entry, click **Copy to clipboard**.

4. Paste and run the command in your terminal.

Verification

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

6.3. ADDITIONAL RESOURCES

- See the Podman documentation for options to use when pulling images.

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**/**jobs**/)

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.