Oracle Linux Automation Manager 2.3

User's Guide for Private Automation Hub

G32926-01
June 2025

Oracle Linux Automation Manager 2.3 User's Guide for Private Automation Hub,

G32926-01

Contents

Preface

Oracle Linux Automation Manager 2.3: Private Automation Hub User's Guide describes how to use Oracle Linux Automation Manager Private Automation Hub.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at https://www.oracle.com/corporate/accessibility/.

Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1

About Private Automation Hub

Private Automation Hub is an Oracle Linux Automation Manager feature that lets you synchronize custom collections and execution environment images for use with Oracle Linux Automation Manager deployments. Private Automation Hub can synchronize collections and execution environments from remote container registries that that you want to host locally.

This guide provides instructions for using Private Automation Hub.

2

Setting Up Permissions for Groups, and Users

This chapter describes how Private Automation Hub enables administrators to create roles, groups, and users where permissions can be allocated at the group level and defined at the role level. These permissions are based on role-based access controls.

> **Note:**
>
> You can integrate the Private Automation Hub access levels discussed in this chapter with external identity management services, such as LDAP. Note that LDAP user account information doesn't appear in Private Automation Hub until after the LDAP user account first logs in to Private Automation Hub. See Oracle Linux Automation Manager 2.3: Private Automation Hub Installation Guide for more information about LDAP authentication and mappings for users and groups.

You can assign roles that specify which permissions are available to a group, and hence available to all the users within that group. You can add from the predefined roles listed in the following table.

> **Note:**
>
> Additionally, you can create custom roles based one or more of the predefined permissions.

Table 2-1 Role-Based Access Control Role Descriptions

Role	Permissions	Description
galaxy.collection_admin	Add namespace Change namespace Delete namespace Upload to namespace Modify Ansible repo content Delete collection Change collection remote View collection remote	Members of a group with this role can do the following: • Create, change, and delete a namespace. • Upload a collection to a namespace. • Use the Approval feature to certify or reject content in the Staging repository and thus move it to the Publishing or Rejected repositories. • Delete collections. • Use the Repository Management feature to Configure remote repositories.

Table 2-1 (Cont.) Role-Based Access Control Role Descriptions

Role	Permissions	Description
galaxy.collection_curator	Modify Ansible repo content Change collection remote View collection remote	Members of a group with this role can do the following: • Use the Approval feature to certify or reject content in the Staging repository and thus move it to the Publishing or Rejected repositories respectively. • Use the Repository Management feature to Configure remote repositories.
galaxy.collection_namespace_owner	Change namespace Upload to namespace	Members of a group with this role can do the following: • Change a namespace. • Upload a collection to a namespace.
galaxy.collection_publisher	Add namespace Change namespace Upload to namespace	Members of a group with this role can do the following: • Create and change a namespace. • Upload a collection to a namespace.
galaxy.content_admin	Add namespace Change namespace Delete namespace Upload to namespace Change collection remote View collection remote Create new containers Change container namespace permissions Change containers Change image tags Push to existing containers Delete container repository Add remote registry Change remote registry Delete remote registry	Members of a group with this role can do the following: • Create, change, and delete a namespace. • Upload a collection to a namespace. • Use the Repository Management feature to Configure remote repositories. • Manage container repositories. • Add, change, or delete remote registries added to Private Automation Hub.

Table 2-1 (Cont.) Role-Based Access Control Role Descriptions

Role	Permissions	Description
galaxy.execution_environment_admin	Create new containers Change container namespace permissions Change containers Change image tags Push to existing containers Delete container repository Add remote registry Change remote registry Delete remote registry	Members of a group with this role can do the following: • Manage container repositories. • Add, change, or delete remote registries added to Private Automation Hub.
galaxy.execution_environment_collaborator	Change containers Change image tags Push to existing containers	Members of a group with this role can do the following: • Change existing execution environments.
galaxy.execution_environment_namespace_owner	Change container namespace permissions Change containers Change image tags	Members of a group with this role can do the following: • Create and update execution environments under existing container namespaces.
galaxy.execution_environment_publisher	Create new containers Change container namespace permissions Change containers Change image tags Push to existing containers	Members of a group with this role can do the following: • Push, and change execution environments.
galaxy.group_admin	Add group Change group Delete group	Members of a group with this role can do the following: • View, add, remove and change groups.
galaxy.task_admin	Change task Delete task View all tasks	Members of a group with this role can do the following: • View, and cancel any task.

Table 2-1 (Cont.) Role-Based Access Control Role Descriptions

Role	Permissions	Description
galaxy.user_admin	Add a standard user Change a standard user Delete a standard user View a standard user **Note:** Only a super user can edit super user accounts. The galaxy.user_admin role's permissions apply to standard users only.	Members of a group with this role can do the following: • View, add, remove and change users.

Table 2-1 (Cont.) Role-Based Access Control Role Descriptions

Role	Permissions	Description
galaxy.collection_remote_owner	View collection remote Add collection remote Change collection remote Delete collection remote Manage remote roles	Members of a group with this role can manage collection remotes.
galaxy.ansible_repository_owner	Modify Ansible repo content Sign collections View Ansible repository Add Ansible repository Change Ansible repository Delete Ansible repository Manage repository roles Repair Ansible repository	Members of a group with this role can manage repositories.

Setting Up Users

Private Automation Hub provides the following user types:

The Default `admin` Super User
When you install Private Automation Hub, a super user with username `admin` is created for you automatically. The `admin` account enables you to log in and set up your system, for example by creating users, other super users, groups, and roles as required by your organization. By default, `admin` does not belong to any group.

> **Note:**
>
> Super users, such as `admin`, have all system permissions regardless of groups they belong to.

Super Users
Private Automation Hub enables you to use a super user account to create other super users in addition to the default **`admin`** user.

Users
Private Automation Hub also enables you to create standard users who do not have super-user privileges.

> **Note:**
>
> **Standard users get most permissions by virtue of their group memberships.** For example, if you create standard user *standard_user_1*, the newly created user will not be able to upload any collections to the namespaces you have in your Private Automation Hub. To enable *standard_user_1* to upload collections to existing namespaces, you would need to carry out additional steps similar to the following:
>
> 1. Create group *Group_Namespace_Uploaders*.
> 2. Assign a built-in role, for example `galaxy.collection_namespace_owner`, that has permissions to upload to a namespace, to group *Group_Namespace_Uploaders*.
> 3. Add `standard_user_1` to group *Group_Namespace_Uploaders*.
> 4. Verify `standard_user_1` can log on and upload collections to namespaces in Private Automation Hub.
>
> For more information on groups and roles see Setting Up Permissions for Groups, and Users, Setting Up Roles, and Setting Up Groups

To set up a user, do the following:

1. Log into Private Automation Hub.
2. From the **User Access** section, click **Users**.

   The Users page appears.
3. Click the **Create** button.

   The Create new user page appears.
4. In the Username field, enter a username.
5. In the First name field, enter a first name.
6. In the Last name field, enter a last name.
7. In the Email field, enter an email address.
8. In the Password field, enter a password.

> **Note:**
>
> The password must contain at least 9 characters, and include special characters , ex <!@$%>. Avoid using common names or expressions.

9. In the Password confirmation field, repeat the password.
10. From the Groups list, select one or more groups.
11. Click the **User type** button if you want the user to have super-user privileges.
12. Click **Save**.

Setting Up Roles

To create custom roles based on permissions associated to the predefined roles, do the following:

1. Log into Private Automation Hub.
2. From the **User Access** section, click **Roles**.

   The Roles page appears listing all available predefined and custom roles.
3. Click the **Add roles** button.

   The Create a new role page appears.
4. In the Name field, enter a role name. The name must begin with the word `galaxy.` and can contain only letters and numbers.
5. In the Description field, enter a description of the role.
6. In the permissions area, select one or more permissions from one or more of the predefined permissions.
7. Click **Save**.
   Your newly created role is added to the list on the Roles page.

Setting Up Groups

To create a group, do the following:

1. Log into Private Automation Hub.
2. From the **User Access** section, click **Groups**.

   The groups page appears.
3. Click the **Create** button.

   The Create a group dialog appears.
4. In the **Name** field, enter a name for your group.
5. Click **Create**.
   A new page for the group appears.
6. Click the **Access** tab.
7. Click **Add roles**.
   The Add roles dialog appears.
8. From the Select roles area, select from the list of roles that define the permissions available to users associated to this group. For more information about the predefined roles, see Setting Up Permissions for Groups, and Users. For more information about custom roles, see Setting Up Roles.
9. Click **Next**.
   The Preview page appears.
10. Click **Add**.
    The group page appears.
11. Click the **Users** tab.
    A list of users associated to the group appears. As this group is newly created, no users are listed.
12. Click **Add**.
    The Add selected users to group dialog appears.
13. From the list, select one or more user.
14. Click **Add**.
    The users you have added now appear in the Users tab.

3
Working with Namespaces and Collections

The Private Automation Hub uses namespaces to organize collections. Working with namespaces requires having a user that's a member of a group with permissions to create, edit and upload collections to namespaces. See Setting Up Permissions for Groups, and Users for more information about groups and permissions to determine how to configure Private Automation Hub for your content curators and developers.

Creating NameSpaces

Before you can upload a collection into Private Automation Hub, you must first create a namespace for the collection. This namespace must match the namespace defined in the collection you want to upload.

Typically, the namespace for a collection represents the company or group that produced the namespace. For example, the `oracle` is the namespace for the Oracle Cloud Infrastructure `OCI` collection. The namespace plus collection format is often found in the name of the collection tar file that you download. However, the tar file naming convention does not always reflect the namespace defined in the collection. To ensure you have the correct namespace, you can download a collection, expand it, and find the MANFEST.json where the namespace is defined.

To create a namespace, do the following:

1. Log into Private Automation Hub.
2. From the **Collections** section, click **Namespaces**.

   The Namespaces page appears.
3. Click the **Create** button.

   The Create a new namespace form appears.
4. In the Name field, enter the name of the namespace for the collection you want to upload.
5. Click **Create**.
   The newly create namespace is displayed in the Collections tab.
6. If you want to associate a group to the namespace, do the following:
   a. Click the **Access** tab.
   b. Click **Select a group**.
      The Select a group dialog appears.
   c. In the Select a group area, select a group.
   d. Click **Next**.
   e. In the Select role(s) area, select from one or more of the available roles relating to managing namespaces.
   f. Click **Next**.
   g. In the Preview area, review the permissions relating to the roles you selected, then click **Add**.
7. Click the `CLI configuration` tab.

A URL for the namespace you have created appears that you can use with the Private Automation Hub CLI to upload new collections to the namespace. By default it has the following format:

```
https://<PAH_Host>/api/galaxy/
```

In the previous example, *<PAH_Host>* is the host name or IP address of the Private Automation Hub server.

8. Click the **Collections** tab.
   The Upload Collections button appears that you can use to upload a manually downloaded collection tar file. For more information, see Uploading Collections.

Uploading Collections

Private Automation Hub enables you to upload collections archived in `tar.gz` files.

Depending upon the `olpah_require_content_approval` setting of your Private Automation Hub instance, an uploaded collection might require approval before it is moved to the published content repository. The possible settings for `olpah_require_content_approval` are as follows:

- `olpah_require_content_approval = False`

  This is the default. No approval is required. Collections are uploaded directly to the published repository and appear under their respective Namespaces in Private Automation Hub.

- `olpah_require_content_approval = True`

  Under this setting, uploaded collections are initially uploaded to the staging repository and appear in the Approval dashboard where a user with the appropriate permissions can approve or reject them. Upon approval, collections are moved to the published repository and appear under their respective Namespaces. Conversely, rejected collections are moved to the rejected repository,

For more information on setting the value of `olpah_require_content_approval`, see Oracle Linux Automation Manager 2.3: Private Automation Hub Installation Guide.

To upload a collection, do the following:

1. Log into Private Automation Hub.
2. From the **Collections** section, click **Namespaces**.

   The Namespaces page appears.
3. Locate the namespace whose name matches the namespace of the collection you intend to upload.

   Click **View Collections.**

   A page displaying published collections for the namespace appears.
4. Click **Upload collection**.

   The New collection page appears.
5. Click inside the Select file box and browse to the `tar.gz` file with the collection you intend to upload.
6. Select one of the following:

- **Staging Repos**: Select this to upload the version of the collection to a staging repository.
- **All Repos**: Select this to upload a version to another repository.

7. Select the file and click **Upload**.

   The My imports page appears displaying the import log so you can follow the progress of the import. The log displays the success or failure of the operation. You can click **Task Management** from the main navigation menu to see the status of the tasks relating to your collection upload.

8. Do one of the following:
   - If you have selected the Staging Repos option, and you have enabled approvals, perform the steps described in Approving Uploaded Collections.
   - If you have selected the All Repos option, see Viewing and Syncing Local Repositories to find the repository where the collection is now located.

Approving Uploaded Collections

If the `olpah_require_content_approval` setting of the Private Automation Hub instance is set to `True`, review the uploaded collections listed in the Approval dashboard and decide whether to approve or reject them.

To approve an uploaded collection in the Approval dashboard, do the following:

1. Log into Private Automation Hub.
2. From the **Collections** section, click **Approval**.

   The Approval dashboard page appears and displays a list of collections in a table.

3. Review the collections with a `Needs review` status in their status column. All collections requiring approval are in the Staging repository or in a repository with the staging pipeline enabled.
4. For each collection to approve, click the **Approve** button.

   The approved collections are moved from the staging to the published repository where users can download and use them.

   > **Note:**
   >
   > If you have more than one repository with the Approved pipeline enabled, a screen appears enabling you to choose which repository to send the approved collection to. For more information about piplines, see Creating a Local Repository.

5. Verify the collections you have approved now appear in the Collections page.
6. You can click **Task Management** from the main navigation menu to see the status of the `move_content` operation that has moved the collection from the staging to the published repository.

Rejecting Uploaded Collections

If the `olpah_require_content_approval` setting of your Private Automation Hub instance is set to `True`, you will need to review the uploaded collections listed in the Approval dashboard and decide whether to approve or reject them.

To reject an uploaded collection in the Approval dashboard, do the following:

1. Log into Private Automation Hub.
2. From the **Collections** section, click **Approval**.

   The Approval dashboard page appears and displays a list of collections in a table.
3. Review the collections with a `Needs review` status in their status column.
4. For each collection you wish to reject, do the following:
   a. Click the Actions button at the end of the row in which the collection is listed.

      A menu appears.
   b. Click the **Reject** option on the menu.

      The rejected collections are moved from the staging to the rejected repository.
5. You can click **Task Management** from the main navigation menu to see the status of the `move_content` operation that has moved the collection from the staging to the rejected repository.

Working With Repositories

This chapter describes how Private Automation Hub enables you to view and manage your local and remote collection repositories.

Viewing and Syncing Local Repositories

To view and sync local repositories, do the following:

1. Log into Private Automation Hub.
2. From the **Collections** section, click **Repositories**.

   The Repositories page appears.

   The table listing the local repositories includes the following columns:

   Repository name
   The name of the repository.

   For more information on the local repositories and their roles see The Purposes of the Different Local Repositories.

   Labels
   Any labels applied to the repository appear in this column. Repository labels can change the context in which a repository is seen. Available options are:

   - **Hide from search**`hide_from_search` This label prevents collections in this repository from appearing on the home page.

- **Pipeline** Valid pipelines options include:
  - **None** Users require permissions to change or upload content to the repository.
  - **Approved** Users can only publish to this repository if the content is approved by an authorized user.
  - **Staging** Any user with upload permissions for a namespace can upload collections to this repository. Users can't view the collection in the search page unless the collection is approved for viewing.
  - **Rejected** These are repositories that aren't approved.

Private
Whether the repository is private or public.

Sync status
The synchronization date shows when the repository was last syncrhonized with a remote repository, if a remote repository is configured. Possible options are:

- The date when the repository was last synchronized.
- **no remote**: This indicates that no remote repository is configured for the repository.
- **never synced**: This indicates the the repository was never synchronized with its remote repository.

Created Date
The date when the repository was created.

3. Select a repository to view.
4. Review the respository information.

   The Distribution URL . It shows the URL to use when uploading and downloading collections to and from a repository from an environment from outside of the Private Automation Hub GUI.

   The Distribution URL has a format similar to the following:

   `https://private_automation_hub/api/galaxy/content/repository_name/`.

> **Note:**
>
> In Oracle Linux Automation Manager, you can configure projects to download collections from this repository distribution URL. For more information see Oracle Linux Automation Manager 2.3: User's Guide and API token management.

5. Click the **Copy CLI Configuration** button to copy the CLI configuration.

   The Successfully copied to clipboard dialog appears containing a sample configuration template to help you configure an `ansible.cfg` file on a host from which you need to run `ansible-galaxy` commands against a repository.

   The contents of the CLI configuration column for repository *repository_name* is similar to the following:

```
[galaxy]
          server_list = published_repo
          [galaxy_server.published_repo]
          url= https://private_automation_hub/api/galaxy/content/
```

```
repository_name/
          token=<put your token here>
```

The token in the preceding example refers to the API token you generate for the Private Automation Hub account, as described in API token management.

6. To sync a repository with a remote repository, do the following:
   a. Click **Sync**.

      The Sync repository dialog appears.
   b. Enable **Mirror** if you want content not present in the remote repository to be removed from the local repository in addition to adding any new content.
   c. Disable **Mirror** if you want the sync to only add new content.
   d. Enable **Optimize** to perform the sync only if changes are detected on the remote repository.
   e. Disable **Optimize** to force a sync regardless of whether changes exist on the remote repository.
   f. Click **Sync**.

> **Note:**
>
> To monitor the progress of the sync operation, you can navigate to the **Task Management** page, find the sync operation in the task list and click the **Task name**. The the page shows the progress of each step as it completes. This can take some time depending on the size and version of collections added to the `requirements.yml` file configured in the Creating a Remote Repository procedure.

The Purposes of the Different Local Repositories

The following list describes the purposes of the different local repositories:

staging
The **staging** repository contains uploaded collections that are awaiting review before being approved or rejected.

> **Note:**
>
> If `olpah_require_content_approval` is set to `False` in your configuration, collections go straight to the **published** repository without any need for approval. See Uploading Collections for more information about the approval process.

published
Once a collection is approved, it is moved to the **published** repository and is available for download.

rejected
Collections that have been reviewed and **rejected** are moved to the rejected repository.

community
The local **community** repository contains collections downloaded from a remote repository as configured by a remote repo connection (also named **community**) on the remote tab of the Repo Management page. By default, the remote community repo connection directs to https://galaxy.ansible.com/api/. For more information see Creating a Remote Repository

Creating a Remote Repository

Private Automation Hub enables you to sync collections from a remote repository down to your local **community** repository by configuring a remote repo connection (also called **community**).

To add a remote repo connection, do the following:

1. Log onto Private Automation Hub.
2. From the **Collections** section, click **Remotes**.

   The Remotes page appears. A table displaying the **community** remote repo connection is displayed.
3. Click **Add Remote**.
   The Add new remote page appears.
4. In the **Name** field, enter a name.
5. In the **URL** field, enter the address of the remote repository you want to download connections from. For example, the default remote community repository points to https://galaxy.ansible.com/api/.
6. (Optional) Enable **Include all dependencies when syncing a collection**, to include all dependencies when syncing a collection.
7. (Optional) In the **Token** field, enter a token for authenticating to the server URL.
8. (Optional) In the **SSO URL** field, enter an SSO URL.
9. (Optional) In the **YAML requirements** field, Click **Upload** and select a `requirements.yml` file that identifies the collections to synchronize from the remote repository. The following provides an example of what a `requirements.yml file` might contain:

```
collections:
  - name: amazon.aws
    source: https://galaxy.ansible.com
    version: 1.2.1
  - name: junipernetworks.junos
    source: https://galaxy.ansible.com
  - name: f5networks.f5_modules
    source: https://galaxy.ansible.com
  - name: oracle.oci
    source: https://galaxy.ansible.com
```

> **Note:**
>
> In the previous example, the version field indicates which version of the collection that Private Automation Hub pulls. When no version field is specified, Private Automation Hub pulls all versions of the collection.

You can also use the YAML editor to directly copy and paste in the contents of the requirements.yml file.

10. (Optional) In the **Username** field, enter a username for connecting to the remote repository.
11. (Optional) In the **Password** field, enter a password for connecting to the remote repository.
12. (Optional) Click **Show advanced options:** if you need to configure proxy server settings or perform any further authentication configuration for a connection to a chosen remote repository.
13. Click **Save**

Creating a Local Repository

To create a local repositories on the system, do the following:

1. Sign in Private Automation Hub.
2. From the **Collections** section, click **Repositories**.

   The Repositories page appears.
3. Click **Add repository**.
   The Add new repository page appears.
4. In the Name field, enter a name for the repository.
5. In the Description field, enter a description for the repository.
6. In the Retained number of versions field, enter the maximum number of versions to retain. Leaving this field blank causes all versions to be retained.
7. To make the repository available to users for sync, download, or search operations, enable **Create a "*repository_name*" distribution** where *repository_name* is the name of the repository you're creating.
8. From the Pipeline dropdown list, select one of the following:
   - **None** Users require permissions to change or upload content to the repository.
   - **Approved** Users can only publish to this repository if the content is approved by an authorized user.
   - **Staging** Any user with upload permissions for a namespace can upload collections to this repository. Users can't view the collection in the search page unless the collection is approved for viewing.
9. To prevent collections in this repository from appearing on the home page, enable **Hide from search**. This is automatically disabled if you selected the Staging pipeline.
10. To make the repository private, enable **Make Private**.
11. To associate the repository with a remote repository, select a remote repository from the Remote drop down list. For more information about adding a remote repository, see Creating a Remote Repository.
12. Click **Save**.

API token management

You can generate an API token for your Private Automation Hub account to enable you to authenticate your connection to the API from outside of the application GUI, for example in one of the following scenarios:

- You might need to run command-line `ansible-galaxy` commands to upload and download collections to and from repositories in Private Automation Hub. In such scenarios you would typically add your API token to your `ansible.cfg` file.
- You might need to set up an Oracle Linux Authentication Manager instance to download collections from a local repository in your Private Automation Hub. In such a scenario you would add your API token to a credential resource in Oracle Linux Authentication Manager.

Providing your user account has the necessary privileges in Private Automation Hub, your account's API token will provide you the access needed in the preceding example scenarios.

To generate an API token for your account, do the following:

> **WARNING:**
>
> Loading a new token will delete your previous token, and any configurations using the previous token will therefore have to be updated with the new token value.

1. Log into Private Automation Hub.
2. From the **Collections** section, click **API token**.

   The API token page appears.
3. Click **Load token** to generate and load a new token to be used for authenticating to Private Automation Hub.
4. The token is generated and displayed.

   > **WARNING:**
   >
   > - Store the API token securely. It protects your content.
   > - The token is displayed once only. The token will never be displayed again.
5. Update any configurations that used your previous account's API token with the new token value.

Accessing Private Automation Hub Collections from Oracle Linux Automation Manager

You can set up Oracle Linux Automation Manager to access ansible collections from the following Private Automation Hub resources:

- **Custom Execution Environments**

  See for Creating Custom Execution Environments more information about creating custom execution environments that contain ansible collections.
- **Collection Repositories**

  See Working With Repositories for more information on working with collection repositories.

The following sections give an overview of how you set up SCM projects in Oracle Linux Automation Manager to access the resources in the preceding list.

Accessing Collections in Private Automation Hub Custom Execution Environments

To access collections in an execution environment in Private Automation Hub, you need to create a resource by the same name in Oracle Linux Automation Manager and set its properties as follows:

- **Name:**

  Choose a name for you execution environment in Oracle Linux Automation Manager, for example, `My_Access_To_Private_Auto_Hub_EE`.

- **Image:**

  The full container image location, including the container registry, image name, and version tag, for example:

  ```
  https://private_automation_hub/my_custom_exe_environment:latest
  ```

- **Organization:**

  Select the organization whose projects need to have access to the custom execution environment referenced in the **Image** property.

- **Registry credential:**

  This is a **Container Registry** type of credential and contains the Private Automation Hub API token that has the necessary access to the custom execution environment you wish to access. This token allows the Oracle Linux Automation Manager credential to authenticate itself to Private Automation Hub.

For more information on creating execution environments and projects in Oracle Linux Automation Manager, see Oracle Linux Automation Manager 2.3: User's Guide

Accessing Collections Contained in Private Automation Hub Repositories

Oracle Linux Automation Manager provides the **Ansible Galaxy/Automation Hub API Token** credential type to set up SCM projects with access to specific repositories in Private Automation Hub. The Ansible Galaxy/Automation Hub API Token credential has the following fields to enable this access to be set up:

- **Galaxy Server URL:**

  This is the URL to the Private Automation Hub repository you wish the SCM project to access its collections from. The URL will be of the following format:

  ```
  https://private_automation_hub/api/galaxy/content/published/
  ```

- **API Token:**

  This is the Private Automation Hub API token that has necessary access to the repository in question. This token allows the Oracle Linux Automation Manager credential to authenticate itself to Private Automation Hub.

You can add one or more such credentials to an organization resource in your Oracle Linux Automation Manager instance. Any projects added to an organization with such credentials will access collections from the locations that the credentials point to.

Note:

The order in which you add credentials to an organization is important:

The order in which you add the credentials to an organization determines the order in which repositories are searched when collections are to be downloaded for a project assigned to that organization.

For more information on Setting up credentials, organizations and projects, see Oracle Linux Automation Manager 2.3: User's Guide

4

Working with Execution Environments

The Private Automation Hub can host execution environments which are containers that you can use with Oracle Linux Automation Manager control and execution plane nodes. Working with execution environment containers requires having a user that is a member of a group with permissions to manage execution environments. See Setting Up Permissions for Groups, and Users for more information about groups and permissions to determine how to configure Private Automation Hub for your content curators and developers.

Configure a Remote Container Registry

Sometimes, Oracle Linux Automation Manager servers can't directly access the Oracle Container Registry to obtain execution environments. In these scenarios, you can install Private Automation Hub in a location accessible to the Oracle Linux Automation Manager and the Oracle Container Registry. You can then configure the Private Automation Hub to periodically download the required execution environments from the Oracle Container Registry so that Oracle Linux Automation Manager can pull them from Private Automation Hub when required.

To configure a connection to a remote container registry, such as the Oracle Container Registry, do the following:

1. Log into Private Automation Hub.
2. From the **Execution Environments** section, click **Remote Registries**.

   The Remote Registries page appears.
3. Click **Add remote registry**.
   The Add remote registry dialogue appears.
4. In the **Name** field, enter a name. For example, Oracle Container Registry.
5. In the **URL** field, enter a base URL for the remote container registry. For example, `https://container-registry.oracle.com/`.
6. In the **Username** field, enter a user name.
7. In the **Password** field, enter a password.
8. Click **Show advanced options**.
9. In the **Proxy URL** field, enter a proxy URL if required by your network.
10. In the **Proxy username** field, enter a user name.
11. In the **Proxy Password** field, enter a password.
12. If required, enable **TLS validation**.
13. In the **Client key** field, enter a PEM encoded private key for TLS authentication.
14. In the **Client certificate** field, enter a PEM encoded client certificate for TLS authentication.
15. From the Download concurrency list, select the total number of simultaneous connections, if required.

16. From the Rate Limit field, enter a limit for the total download rate in requests per second.
17. Click **Save**.

Creating an Execution Environment with Remote Registry

To create an execution environment created with a remote container registry, do the following:

1. Log into Private Automation Hub.
2. From the **Execution Environments** section, click **Execution Environments**.

   The Execution Environments page appears.
3. Click **Add execution environments**.
   The Add execution environment appears.
4. In the **Name** field, enter an execution environment name to represent the container image. Container names can only contain alphanumeric characters, ".", "_", "-" and a up to one "/". For example, `olamprivatehub-ee`.
5. In the **Upstream name** field, the namespace/name format for containers that use a namespace. Use the library/name format with containers that use a library. For example, `oracle_linux_automation_manager/olam-ee`.
6. From the Registry list, select a remote registry that you have configured. For more information, see Configure a Remote Container Registry.
7. In the add tag(s) to include, enter the name of the tag you want to include from the remote registry. For example, `latest`.
8. Click **Add**.
   The added tag appears in the Currently included tags list.
9. In the add tag(s) to exclude, enter the name of the tag you want to exclude from the remote registry. For example, `dev`.
10. Click **Add**.
    The added tag appears in the Currently excluded tags list.
11. Click **Save**.
    The new execution environment appears in the Execution Environment page.

Synchronizing an Execution Environment from a Remote Registry

To synchronize an execution environment from a remote registry, do the following:

1. Log into Private Automation Hub.
2. From the **Execution Environments** section, click **Execution Environments**.

   The Execution Environments page appears.
3. Click the menu button on the container you want to synchronize.
4. Click **Sync from registry**.
   An info alert dialogue box appears stating `Sync started for execution environment "<container_name>". See the task management detail page for the status of this task.` If a new version of the container exists, the new version is uploaded and the Last Modified field changes to the present time.

View Execution Environment Details

To view execution environment details, do the following:

1. Log into Private Automation Hub.
2. From the **Execution Environments** section, click **Execution Environments**.

   The Execution Environments page appears.
3. Click the container repository name.
   The Detail tab appears with the command to pull the image using Podman.
4. Click the **Activity** tab.
   A list of changes and the dates when the changes occurred appear.
5. Click the **Images** tab.
   Information about different versions of the image appears. You can add or edit tags or delete the image.
6. Click the **Access** tab.
7. If you want to give users access, click **Select a user**.
   The Select a user dialog appears.
8. From the list, select a user.
9. Click **Next**.
10. From the list, select one or more roles to apply to the user.
11. Click **Next**.
12. View the preview of your selections, then click **Add**.
13. If you want associate the execution environment to a new group, click **Select a group**.
    The Select a group dialog appears.
14. From the list, select a group.
15. Click Next.
16. From the list, select one or more roles.
    The preview area appears.
17. Click **Add**.

5

Creating Custom Execution Environments

The builder utility allows you to customize and create execution environments that you can upload to Private Automation Hub where Oracle Linux Automation Manager can access them and use them to run playbooks. Being able to use customized container images as execution environments to run playbooks allows you to ensure you have all the packages and dependencies you need on the container image necessary to run playbooks in a consistent and dependable way.

> **Note:**
>
> While the Builder utility enables you to create custom execution environments, in the case of an issue with the custom execution environment, Oracle support may ask you to revert to the Oracle provided OLAM-EE default image to troubleshoot the problem.

Execution environments that you customize with the Builder utility contains:

- The base OLAM-EE container image accessible from the Oracle Container Registry. Two versions of this container image are available where the main version includes the OCI SDK and the Ansible OCI Collection and the minimal version does not. Because the Ansible OCI Collection is updated on a more regular cadence to take advantage of new OCI features, the main version of thie OLAM-EE container image releases new more frequently updates.
- A special builder image required for images created with the Builder utility that's also accessible from the Oracle Container Registry. The Builder utility uses this image when compiling container images.
- One or more ansible collections.
- Python or system dependencies (for example, modules, plugins in collections, custom scripts, bash commands), or a combination of both.

You can use the Builder utility to create execution environments with format version 1, 2, or 3. For examples, see Configuring a Custom Execution Environment Using Format 1, Configuring a Custom Execution Environment Using Format 2, and Configuring a Custom Execution Environment Using Format 3.

The default `olam-ee` available on the Oracle Container Registry uses the latest version of the Oracle Linux 8 container image. When building custom execution environments, we always use the latest version available of `ansible-core` and its python dependencies for the Oracle Linux 8 release provided. For more information about `olam-ee` and Python dependencies, see Oracle Linux Automation Manager 2.3: Release Notes.

In some cases, you might need a different version of Python not available on the default execution environment to run job templates for the playbook to work. To change the python version used by the playbook, apply the `ansible_python_interpreter` variable in the Oracle Linux Automation Manager UI. For example,

```
"ansible_python_interpreter: /usr/bin/python<version>"
```

In the previous example, `version` is the python version the playbook needs.

For more information about setting variables on job templates, see the Oracle Linux Automation Manager 2.3: User's Guide.

Configuring a Custom Execution Environment Using Format 1

To configure a custom execution environment using format 1, do the following:

1. On the host where you have installed the Builder utility, create a working directory for Builder utility. For example,

   ```
   mkdir ~/builder-utility
   cd ~/builder-utility
   ```

2. Create the following files:

   ```
   touch ansible.cfg execution-environment.yml requirements.yml
   requirements.txt bindep.txt
   ```

3. In the execution-environment.yml file, add the following parameters:

   ```
   ---
   version: 1
   build_arg_defaults:
     EE_BASE_IMAGE: 'container-registry.oracle.com/
   oracle_linux_automation_manager/olam-ee:2.3-<OS_Version>'
     EE_BUILDER_IMAGE: 'container-registry.oracle.com/
   oracle_linux_automation_manager/olam-builder:2.3-<OS_Version>'
     ANSIBLE_GALAXY_CLI_COLLECTION_OPTS: "--ignore-certs"
   ansible_config: 'ansible.cfg'
   dependencies:
     galaxy: requirements.yml
     python: requirements.txt
     system: bindep.txt
   additional_build_steps:
     prepend: |
       RUN whoami
       RUN cat /etc/os-release
     append:
       - RUN echo This is a post-install command!
       - RUN ls -la /etc
   ```

   In the previous example,

   - **EE_BASE_IMAGE**: This parameter requires the base olam-ee, which you can obtain from `container-registry.oracle.com/oracle_linux_automation_manager/olam-ee:2.3-<OS_version>`where *OS_version* is `ol8` or `ol9` for Oracle Linux 8 or Oracle Linux 9. Alternatively, you can also obtain a copy of this image from Private Automation Hub if you have configured Private Automation Hub for this purpose.

> **Note:**
>
> Two versions of this container image are available on the Oracle Container Registry where the main version includes the OCI SDK and the Ansible OCI Collection and the minimal version does not. The version specified in the previous example uses the main version. For more information about the minimal version, see Oracle Linux Automation Manager 2.3: Release Notes.

- **EE_BUILDER_IMAGE**: This parameter requires the olam-builder image, which you can obtain from `container-registry.oracle.com/oracle_linux_automation_manager/olam-builder-<OS_version>` where *OS_version* is `ol8` or `ol9` for Oracle Linux 8 or Oracle Linux 9. The builder container image contains build libraries necessary to create customer execution environments. Alternatively, you can also obtain a copy of this image from Private Automation Hub if you have configured Private Automation Hub for this purpose.
- **ANSIBLE_GALAXY_CLI_COLLECTION_OPTS**: Use the `"--ignore-certs"` option. This parameter allows you to obtain an image from a repository without using self signed certificates.
- **ansible_config**: You can specify the `ansible.cfg` file to pass a token and other settings for a private account to a Private Automation Hub server. Otherwise, the Builder utility uses ansible.galaxy to download Ansible collections. Listing the config file path as a string includes it as a build argument in the initial phase of the build.
- **dependencies**: This section is a dictionary value that defines the Ansible Galaxy, Python, and system dependencies that must be installed into the final container.

> **Note:**
>
> Any dependencies listed in this section that are outside of the Oracle Linux or Oracle Linux Automation Manager scope of coverage are outside of the scope of coverage for support.

Valid keys for this section are as follows:

  - **galaxy**: This parameter is the path to a file that defines the collection dependencies. The Builder utility installs these with the `ansible-galaxy collection install -r requirements.yml` command. The supplied value may be a relative path from the directory of the execution environment definition's folder, or an absolute path.
    The format for specifying a collection in the requirements.yml file can be as follows:

    ```
    ---
    collections:
      - name: <namespace>.<collection_name>
    ```

    In the previous example, *<namespace>* is the namespace of the collection (for example, `oracle`) and *<collection_name>* is the name of the collection to be installed (for example, `oci`). You can repeat this combination for each collection you want to use.

You can also indicate a specific version, source type, and source location as follows:

```
---
collections:
- name <namespace>.<collection_name>
  version: <version_number or "<range_identifyer><version_number>"
  type: <source_type>
```

In the previous example, <version_number> can be any collection version number. When specified with one or more <range_identifyer>, then the version number is always the most recent version of the collection possible within the range specified. For example, the following indicates that the version should be greater than or equal to 5.0.0, but less than 6.0.0:

```
---
collections:
- name: oracle.oci
  version: ">=5.0.0,<6.0.0"
  type: galaxy
```

You can use the following range identifiers:

* *
  The most recent version. This is the default.
* !=
  Not equal to the version specified.
* ==
  Exactly the version specified.
* >=
  Greater than or equal to the version specified.
* >
  Greater than the version specified.
* <=
  Less than or equal to the version specified.
* <
  Less than the version specified.

– **python**: This string value is the path to a file containing the Python dependencies to be installed with the `pip install -r requirements.txt` command. The supplied value may be a relative path from the directory of the execution environment definition's folder, or an absolute path.
The format for specifying a python dependency in the requirements.txt file can be as follows:

```
<someproject>
<someproject><version_specifier><version_number>
<someproject><version_specifier><version_number>,<version_specifier>
<version_number>
```

In the previous example, <*someproject*> is the name of the project. You can specify the project name alone, which find the latest version of that project, or you can specify a project with one or more <*version_identifier*> and <*verion_number*> combinations to return a specific version of the project you want to install. For example:

```
requests>=2.4.2
xmltodict
azure-cli-core==2.11.1
```

For more information about available version specifiers, see https://peps.python.org/pep-0440/#version-specifiers.

- **system**: This string value is points to a bindep.txt requirements file. This will be processed by bindep and then passed to dnf, other platforms are not yet supported. For example:

```
gcc
libcurl-devel
libxml2-devel
python39-devel
openssl-devel
```

> **Note:**
>
> Some collections require different versions of Python to be installed. For example, the `ovirt.ovirt` collection requires `python3.9-devel` when installed on Oracle Linux 9 and python3.11-devel or possibly python3.12-devel when installed on Oracle Linux 8.

For more information about writing a system requirements file and optionally specifying version constraints, see https://docs.opendev.org/opendev/bindep/latest/readme.html#writing-requirements-files.

- **additional_build_steps**: Additional commands may be specified in the additional_build_steps section, either for before the main build steps (prepend) or after (append). The syntax needs to be one of the following:
  - a multi-line string (example shown in the prepend section above)
  - a list (as shown via append)

4. Save the file.
5. Complete the ansible.cfg and any dependency files as required.
6. Run the following command from your working directory:

```
ansible-builder build -t <repository:tag>
```

In the previous example, <repository:tag> is the repository and tag of the custom execution environment. For example, <*hostname.domain*>/custom_ee:latest.

> **Note:**
>
> A warning message appears indicating that format 1 is deprecated. Consider using format 3 instead. For more information, see Configuring a Custom Execution Environment Using Format 3.

> **Note:**
>
> For more information as the builder utility processes, use the -v 3 option at the end of the command provides the most information while 1 is the least. For example,
>
> ```
> ansible-builder build -v 3
> ```

7. Verify that the images have been created. Run the following Podman command:

   ```
   podman images
   ```

Configuring a Custom Execution Environment Using Format 2

To configure a custom execution environment using format 2, do the following:

1. On the host where you have installed the Builder utility, create a working directory for Builder utility. For example,

   ```
   mkdir ~/builder-utility
   cd ~/builder-utility
   ```

2. Create the following files:

   ```
   touch ansible.cfg execution-environment.yml requirements.yml
   requirements.txt bindep.txt
   ```

3. In the execution-environment.yml file, add the following parameters:

   ```
   ---
   version: 2

   build_arg_defaults:
    ANSIBLE_GALAXY_CLI_COLLECTION_OPTS: "--ignore-certs"

   ansible_config: 'ansible.cfg'

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

   additional_build_steps:
     prepend: |
   ```

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

images:
 base_image:
  name: container-registry.oracle.com/oracle_linux_automation_manager/olam-
ee:2.3-<OS_version>
 builder_image:
  name: container-registry.oracle.com/oracle_linux_automation_manager/olam-
builder:2.3-<OS_version>
```

In the previous example,

- **ANSIBLE_GALAXY_CLI_COLLECTION_OPTS**: Use the `"--ignore-certs"` option.This parameter allows you to obtain an image from a repository without using self signed certificates.
- **ansible_config**: You can specify the `ansible.cfg` file to pass a token and other settings for a Private Automation Hub server. Listing the config file path as a string includes it as a build argument in the initial phase of the build.
- **dependencies**: This section is a dictionary value that is defines the Ansible Galaxy, Python, and system dependencies that must be installed into the final container. Valid keys for this section are as follows:
  - **galaxy**: This parameter is the path to a file that defines the collection dependencies. The Builder utility installs these with the `ansible-galaxy collection install -r requirements.yml` command. The supplied value may be a relative path from the directory of the execution environment definition's folder, or an absolute path.
    The format for specifying a collection in the requirements.yml file can be as follows:

    ```
    ---
    collections:
    - name <namespace>.<collection_name>
    ```

    In the previous example, *<namespace>* is the namespace of the collection (for example, `oracle`) and *<collection_name>* is the name of the collection to be installed (for example, `oci`). You can repeat this combination for each collection you want to use.

    You can also indicate a specific version, source type, and source location as follows:

    ```
    ---
    collections:
      - <namespace>.<collection_name>
          version: <version_number or
    "<range_identifyer><version_number>"
          type: <source_type>
    ```

In the previous example, <version_number> can be any collection version number. When specified with one or more <range_identifyer>, then the version number is always the most recent version of the collection possible within the range specified. For example, the following indicates that the version should be greater than or equal to 5.0.0, but less than 6.0.0:

```
---
collections:
- name: oracle.oci
  version: ">=5.0.0,<6.0.0"
  type: galaxy
```

You can use the following range identifiers:

* *
  The most recent version. This is the default.
* !=
  Not equal to the version specified.
* ==
  Exactly the version specified.
* >=
  Greater than or equal to the version specified.
* >
  Greater than the version specified.
* <=
  Less than or equal to the version specified.
* <
  Less than the version specified.

– **python**: This string value is the path to a file containing the Python dependencies to be installed with the pip install -r ... command. The supplied value may be a relative path from the directory of the execution environment definition's folder, or an absolute path.
The format for specifying a python dependency in the requirements.txt file can be as follows:

```
<someproject>
<someproject><version_specifier><version_number>
<someproject><version_specifier><version_number>,<version_specifier>
<version_number>
```

In the previous example, <*someproject*> is the name of the project. You can specify the project name alone, which find the latest version of that project, or you can specify a project with one or more <*version_identifier*> and <*verion_number*> combinations to return a specific version of the project you want to install. For example:

```
requests>=2.4.2
xmltodict
azure-cli-core==2.11.1
```

For more information about available version specifiers, see https://peps.python.org/pep-0440/#version-specifiers.

- **system**: This string value is points to a `bindep.txt` requirements file. This will be processed by bindep and then passed to dnf, other platforms are not yet supported. For example:

```
gcc
libcurl-devel
libxml2-devel
python39-devel
openssl-devel
```

> **Note:**
>
> Some collections require different versions of Python to be installed. For example, the `ovirt.ovirt` collection requires `python3.9-devel` when installed on Oracle Linux 9 and python3.11-devel or possibly python3.12-devel when installed on Oracle Linux 8.

For more information about writing a system requirements file and optionally specifying version constraints, see https://docs.opendev.org/opendev/bindep/latest/readme.html#writing-requirements-files.

- **additional_build_steps**: Additional commands may be specified in the additional_build_steps section, either for before the main build steps (prepend) or after (append). The syntax needs to be one of the following:
  - a multi-line string (example shown in the prepend section above)
  - a list (as shown via append)
- images: You can use the images key to define base and builder images. With the version 2 format, an execution environment definition may specify a base and builder container image whose signature must be validated before builder will build the resulting image, based on the value of the --container-policy CLI option. Key options are as follows:
  - **base_image**: This parameter requires the base olam-ee, which you can obtain from `container-registry.oracle.com/oracle_linux_automation_manager/olam-ee:2.3-<OS_version>` where *OS_version* is `ol8` or `ol9` for Oracle Linux 8 or Oracle Linux 9. Alternatively, you can also obtain a copy of this image from Private Automation Hub if you have configured Private Automation Hub for this purpose.

> **Note:**
>
> Two versions of this container image are available on the Oracle Container Registry where the main version includes the OCI SDK and the Ansible OCI Collection and the minimal version does not. The version specified in the previous example uses the main version. For more information about the minimal version, see Oracle Linux Automation Manager 2.3: Release Notes.

  - **builder_image**: This parameter requires the olam-builder image, which you can obtain from `container-registry.oracle.com/`

`oracle_linux_automation_manager/olam-builder:2.3-<OS_version>` where *OS_version* is `ol8` or `ol9` for Oracle Linux 8 or Oracle Linux 9. The builder container image contains build libraries necessary to create customer execution environments. Alternatively, you can also obtain a copy of this image from Private Automation Hub if you have configured Private Automation Hub for this purpose.

4. Save the file.
5. Complete the ansible.cfg and any dependency files as required.
6. Run the following command from your working directory:

```
ansible-builder build -t <repository:tag>
```

In the previous example, <repository:tag> is the repository and tag of the custom execution environment. For example, <*hostname.domain*>/custom_ee:latest.

> **Note:**
>
> A warning message appears indicating that format 2 is deprecated. Consider using format 3 insteadl. For more information, see Configuring a Custom Execution Environment Using Format 3.

> **Note:**
>
> For more information as the builder utility processes, use the -v 3 option at the end of the command provides the most information while 1 is the least. For example,
>
> ```
> ansible-builder build -v 3
> ```

7. Verify that the images have been created. Run the following Podman command:

```
podman images
```

Configuring a Custom Execution Environment Using Format 3

To configure a custom execution environment using format 3, do the following:

1. On the host where you have installed the Builder utility, create a working directory for Builder utility. For example,

```
mkdir ~/builder-utility
cd ~/builder-utility
```

2. Create the following files:

```
touch ansible.cfg execution-environment.yml requirements.yml
requirements.txt bindep.txt
```

3. In the execution-environment.yml file, add the following parameters:

```
---
version: 3

build_arg_defaults:
  ANSIBLE_GALAXY_CLI_COLLECTION_OPTS: '--pre --ignore-certs'


dependencies:
  ansible_runner:
    package_pip: ansible-runner
  galaxy: requirements.yml
  python: requirements.txt
  system: bindep.txt

images:
  base_image:
    name: container-registry.oracle.com/oracle_linux_automation_manager/
olam-ee:2.3-<OS_version>

additional_build_files:
    - src: ansible.cfg
      dest: configs

additional_build_steps:
  prepend_galaxy:
    - COPY _build/configs/ansible.cfg /etc/ansible/ansible.cfg
  prepend_final: |
    RUN whoami
    RUN cat /etc/os-release
  append_final:
    - RUN echo This is a post-install command!
    - RUN ls -la /etc
```

In the previous example,

- **ANSIBLE_GALAXY_CLI_COLLECTION_OPTS**: Use the `"--ignore-certs"` option.This parameter allows you to obtain an image from a repository without using self signed certificates.
- **dependencies**: This section is a dictionary value that is defines the Ansible Galaxy, Python, and system dependencies that must be installed into the final container. Valid keys for this section are as follows:
  - **ansible_runner: package_pip: ansible-runner**: During the base build stage, the `ansible-runner` component is customized to be installed by the `package_pip` Python installer. Ephemeral copies of the resulting customized base image are used as the base for all other build stages.
  - **galaxy**: This parameter is the path to a file that defines the collection dependencies. The Builder utility installs these with the `ansible-galaxy collection install -r requirements.yml` command. The supplied value may be a relative path from the directory of the execution environment definition's folder, or an absolute path.

The format for specifying a collection in the requirements.yml file can be as follows:

```
---
collections:
- name <namespace>.<collection_name>
```

In the previous example, *<namespace>* is the namespace of the collection (for example, `oracle`) and *<collection_name>* is the name of the collection to be installed (for example, `oci`). You can repeat this combination for each collection you want to use.

You can also indicate a specific version, source type, and source location as follows:

```
---
collections:
  - name: <namespace>.<collection_name>
     version: <version_number or
"<range_identifyer><version_number>"
     type: <source_type>
```

In the previous example, <version_number> can be any collection version number. When specified with one or more <range_identifyer>, then the version number is always the most recent version of the collection possible within the range specified. For example, the following indicates that the version should be greater than or equal to 5.0.0, but less than 6.0.0:

```
---
collections:
- name: oracle.oci
  version: ">=5.0.0,<6.0.0"
  type: galaxy
- name: ovirt.ovirt
```

You can use the following range identifiers:

* * The most recent version. This is the default.
* != Not equal to the version specified.
* == Exactly the version specified.
* >= Greater than or equal to the version specified.
* > Greater than the version specified.
* <= Less than or equal to the version specified.
* < Less than the version specified.

– **python**: This string value is the path to a file containing the Python dependencies to be installed with the pip install -r ... command. The supplied value may be a relative path from the directory of the execution environment definition's folder, or an absolute path.

The format for specifying a python dependency in the requirements.yml file can be as follows:

```
<someproject>
<someproject><version_specifier><version_number>
<someproject><version_specifier><version_number>,<version_specifier>
<version_number>
```

In the previous example, *<someproject>* is the name of the project. You can specify the project name alone, which find the latest version of that project, or you can specify a project with one or more *<version_identifier>* and *<verion_number>* combinations to return a specific version of the project you want to install. For example:

```
requests>=2.4.2
xmltodict
azure-cli-core==2.11.1
```

For more information about available version specifiers, see https://peps.python.org/pep-0440/#version-specifiers.

– **system**: This string value is points to a `bindep.txt` requirements file. This is processed by bindep and then passed to dnf, other platforms are not yet supported. For example:

```
gcc
libcurl-devel
libxml2-devel
python39-devel
openssl-devel
```

> **Note:**
>
> Some collections require different versions of Python to be installed. For example, the `ovirt.ovirt` collection requires `python3.9-devel` when installed on Oracle Linux 9 and python3.11-devel or possibly python3.12-devel when installed on Oracle Linux 8.

For more information about writing a system requirements file and optionally specifying version constraints, see https://docs.opendev.org/opendev/bindep/latest/readme.html#writing-requirements-files.

- images: You can use the images key to define a base image. With the version 3 format, an execution environment definition may specify a base container image whose signature must be validated before builder will build the resulting image, based on the value of the --container-policy CLI option. The key options is **base_image**. This parameter requires the base olam-ee, which you can obtain from `container-registry.oracle.com/oracle_linux_automation_manager/olam-ee:2.3-<OS_version>` where *OS_version* is `ol8` or `ol9` for Oracle Linux 8 or Oracle Linux 9. Alternatively, you can also obtain a copy of this image from Private Automation Hub if you have configured Private Automation Hub for this purpose.

> **Note:**
>
> Two versions of this container image are available on the Oracle Container Registry where the main version includes the OCI SDK and the Ansible OCI Collection and the minimal version does not. The version specified in the previous example uses the main version. For more information about the minimal version, see Oracle Linux Automation Manager 2.3: Release Notes.

- **additional_build_files**: You can specify a `ansible.cfg` file to pass a token and other settings for a Private Automation Hub server. This file is referenced using the following format:

```
additional_build_files:
    - src: ansible.cfg
      dest: configs
```

> **Note:**
>
> If you omitting specifying the credentials for an Private Automation Hub in the `ansible.cfg` file, then Private Automation Hub defaults to the public https://galaxy.ansible.com.

- **additional_build_steps**: Additional commands may be specified in the additional_build_steps section, either for before the build stages (prepend) or after (append). These stages include the following:
  Base Build Stage: Customizes the specified base image with required components such as Python, pip, ansible-core, and ansible-runner. Validates the presence of these components. Creates ephemeral copies of the customized base image for subsequent stages. Steps include:
  - `prepend_base`: Insert commands before customizing the base image.
  - `append_base`: Insert commands after customizing the base image.

  Galaxy Build Stage: Downloads and stores collections specified in the definition file. Collects Python and system dependencies declared by the collections. Steps include:
  - `prepend_galaxy`: Insert commands before downloading collections.
  - `append_galaxy`: Insert commands after downloading collections.

  Builder Build Stage: Merges Python dependencies from collections and the definition file. Downloads and builds Python dependencies as wheels. Stores the wheels for later installation. Steps include:
  - `prepend_builder`: Insert commands before merging Python dependencies.
  - `append_builder`: Insert commands after building Python wheels.

  Final Build Stage: Installs previously-downloaded collections. Installs system packages and Python packages built during the builder stage. Steps include:
  - `prepend_final`: Insert commands before installing collections and packages.
  - `append_final`: Insert commands after installing collections and packages.

  By injecting custom commands at each stage, you can tailor the image creation process to meet specific requirements or integrate with other tools and systems.

4. Save the file.
5. Complete the `ansible.cfg` and any dependency files as required.
6. Run the following command from your working directory:

```
ansible-builder build -t <repository:tag>
```

In the previous example, <repository:tag> is the repository and tag of the custom execution environment. For example, <*hostname.domain*>/custom_ee:latest.

> **Note:**
>
> For more information as the builder utility processes, use the -v 3 option at the end of the command provides the most information while 1 is the least. For example,
>
> ```
> ansible-builder build -v 3
> ```

7. Verify that the images have been created. Run the following Podman command:

```
podman images
```

Uploading a Custom Execution Environment

To upload a custom execution environment, do the following:

1. From the host where you created the custom execution environment, use Podman to log into the Private Automation Hub instance you want to upload your custom execution environment to. For example,

```
podman login <ip address or hostname> -u admin -p <password> --tls-
verify=false
```

In the previous example, <ip address or hostname> is the IP address or host name of the Private Automation Hub instance and <password> is the password for the admin user on the Private Automation Hub instance.

2. Identify which custom image you want to upload. For example,

```
podman images
```

3. Push the image to your Private Automation Hub. For example,

```
podman push <podman_image_id or repository:tag> <ip address or hostname/
image_name:tag> --tls-verify=false
```

In the previous example, <podman_image_id or repository:tag> is the image ID or the repository and tag of the custom execution environment you want to push and <ip address or hostname/image_name:tag> is the IP address or host name followed by the image name and the tag of the Private Automation Hub instance. For example, the following uses

a podman repository name and tag followed by the hostname, image name, and tag to use on the private automation hub instance:

```
podman push localhost/custom-image:2.1.3 private_automation_hub_host/
custom-image:2.1.3 --tls-verify=false
```

The following uses a podman image id followed by the hostname, image name, and tag to use on the private automation hub instance:

```
podman push 330928308c01 private_automation_hub_host/custom-image:2.1.3 --
tls-verify=false
```

4. Verify that the execution environment now exists on your Private Automation Hub instance. For more information about this task, see View Execution Environment Details.

6
Working with the Command-Line Interface

You can use the command-line interface to manage collections and ansible roles.

Installing the Command-Line Interface

The Command-Line interface is installed on the Private Automation Hub deployment host when you install the Private Automation Hub installer. For more information about installing Private Automation Hub, see Oracle Linux Automation Manager 2.3: Private Automation Hub Installation Guide. You can also install the CLI alone on a separate system rather than use the CLI on the deployment host.

To install a the CLI alone on a separate system, do the following:

1. Use the dnf config-manager tool to enable the yum repositories and do one of the following:
   - If you are using ol8_UEK6, use the following command:

     ```
     sudo dnf config-manager --enable ol8_UEKR6 ol8_appstream
     ```

   - If you are using ol8_UEK7, use the following command:

     ```
     sudo dnf config-manager --enable ol8_UEKR7 ol8_appstream
     ```

2. Ensure that no version of ansible is present on the system. If any are, uninstall them. For example, the following shows that there are no versions of ansible installed:

   ```
   rpm -q ansible
   package ansible is not installed
   ```

3. Install the CLI:

   ```
   sudo dnf install ansible-core
   ```

Using the Command-Line Interface

You interact with the CLI by entering commands with a series of options. The CLI command with the --help flag returns the following syntax information:

```
ansible-galaxy --help
usage: ansible-galaxy [-h] [--version] [-v] TYPE ...

Perform various Role and Collection related operations.

positional arguments:
  TYPE
    collection   Manage an Ansible Galaxy collection.
    role         Manage an Ansible Galaxy role.
```

```
options:
  --version      show program's version number, config file location,
configured module search path, module location, executable location and exit
  -h, --help     show this help message and exit
  -v, --verbose  Causes Ansible to print more debug messages. Adding multiple
-v will increase the verbosity, the builtin plugins currently evaluate up to -
vvvvvv. A
                 reasonable level to start is -vvv, connection debugging
might require -vvvv.
```

The **positional arguments** section lists the available resources to manage using the CLI. You can obtain additional information about these resources by adding the resource name before the --help flag. For example, the following shows actions available for the collection resource:

```
ansible-galaxy collection --help
usage: ansible-galaxy collection [-h] COLLECTION_ACTION ...

positional arguments:
  COLLECTION_ACTION
    download         Download collections and their dependencies as a tarball
                     for an offline install.
    init             Initialize new collection with the base structure of a
                     collection.
    build            Build an Ansible collection artifact that can be
                     published to Ansible Galaxy.
    publish          Publish a collection artifact to Ansible Galaxy.
    install          Install collection(s) from file(s), URL(s) or Ansible
                     Galaxy
    list             Show the name and version of each collection installed in
                     the collections_path.
    verify           Compare checksums with the collection(s) found on the
                     server and the installed copy. This does not verify
                     dependencies.

options:
  -h, --help         show this help message and exit
```

The following shows actions available for the role resource:

```
ansible-galaxy role --help
usage: ansible-galaxy role [-h] ROLE_ACTION ...

positional arguments:
  ROLE_ACTION
    init       Initialize new role with the base structure of a role.
    remove     Delete roles from roles_path.
    delete     Removes the role from Galaxy. It does not remove or alter the
actual GitHub repository.
    list       Show the name and version of each role installed in the
roles_path.
    search     Search the Galaxy database by tags, platforms, author and
multiple keywords.
    import     Import a role into a galaxy server
    setup      Manage the integration between Galaxy and the given source.
```

```
    info       View more details about a specific role.
    install    Install role(s) from file(s), URL(s) or Ansible Galaxy

options:
  -h, --help   show this help message and exit
```

In a similar way, you can use the --help flag to obtain more information about the available actions.

For example, the following command creates a working `ansible.cfg` file :

```
sudo bash -c "cat <<EOF >> /etc/ansible/ansible.cfg
[galaxy]
server_list = release_galaxy

[galaxy_server.release_galaxy]
url=https://galaxy.ansible.com/
EOF"
```

You can now run commands to download collections. For example, the following commands download the `OCI` and `posix` collections:

```
ansible-galaxy collection download ansible.posix -c -s release_galaxy
ansible-galaxy collection download oracle.oci -c -s release_galaxy
```