



# Red Hat Ansible Automation Platform 2.2

## 実行環境の作成および消費

Red Hat Ansible Automation Platform 用の一貫性のある再現可能な自動化実行環境  
を作成します。



## Red Hat Ansible Automation Platform 2.2 実行環境の作成および消費

---

Red Hat Ansible Automation Platform 用の一貫性のある再現可能な自動化実行環境を作成します。

## 法律上の通知

Copyright © 2023 Red Hat, Inc.

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

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

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

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

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

Linux<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

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

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

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

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

All other trademarks are the property of their respective owners.

## 概要

フィードバックの提供: このドキュメントを改善するための提案がある場合、またはエラーを見つけた場合は、テクニカルサポート () に連絡し、Docs コンポーネントを使用して Ansible Automation Platform Jira プロジェクトで issue を作成してください。

---

目次

はじめに .....	3
多様性を受け入れるオープンソースの強化 .....	4
第1章 自動化実行環境の概要 .....	5
1.1. 自動化実行環境について .....	5
第2章 ANSIBLE BUILDER の使用 .....	6
2.1. ANSIBLE BUILDER を使用する理由 .....	6
2.2. ANSIBLE BUILDER のインストール .....	6
2.3. 定義ファイルの構築 .....	6
2.4. ビルドの実行およびコマンドの作成 .....	7
2.5. 定義ファイルの内容の内訳 .....	7
2.6. 任意のビルドコマンド引数 .....	11
2.7. CONTAINERFILE .....	11
2.8. イメージの構築なしで CONTAINERFILE の作成 .....	12
第3章 自動化実行環境の公開 .....	13
3.1. 既存の自動化実行環境イメージのカスタマイズ .....	13
第4章 プライベート自動化ハブコンテナレジストリーへの入力 .....	16
4.1. 前提条件 .....	16
4.2. 自動化ハブで使用するイメージの取得 .....	16
4.3. 自動化ハブで使用するイメージのタグ付け .....	16
4.4. プライベート自動化ハブへのコンテナイメージのプッシュ .....	17
第5章 コンテナレジストリーの設定 .....	19
5.1. 前提条件 .....	19
5.2. コンテナレジストリーへの README の追加 .....	19
5.3. コンテナレジストリーへのアクセスの提供 .....	19
5.4. コンテナイメージのタグ付け .....	20
第6章 コンテナレジストリーからのイメージのプル .....	21
6.1. 前提条件 .....	21
6.2. イメージのプル .....	21
6.3. 関連情報 .....	21
付録A 自動化実行環境の優先順位 .....	22



## はじめに

Ansible Builder を使用して、Red Hat Ansible Automation Platform のニーズに合わせて一貫性のある再現可能な自動化実行環境を作成します。

## 多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、[Red Hat CTO である Chris Wright のメッセージ](#)をご覧ください。

## 第1章 自動化実行環境の概要

各ノードにパッケージをインストールして、ホストシステムにインストールされている他のソフトウェアと対話し、同期を維持する必要があるため、デフォルト以外の依存関係に依存する Ansible コンテンツを使用することは複雑になる可能性があります。

自動化実行環境は、このプロセスを単純化し、Ansible Builder で簡単に作成できます。

### 1.1. 自動化実行環境について

自動化実行環境は、Red Hat Ansible Automation Platform のすべての自動化が実行するコンテナイメージです。自動化実行環境は、自動化の依存関係を通信するための共通の言語を作成し、自動化環境を構築して配布するための標準的な方法を提供します。

自動化実行環境には、以下が含まれます。

- Ansible 2.9 または Ansible Core 2.11-2.13
- Python 3.8-3.10
- Ansible Runner
- Ansible コンテンツコレクション
- コレクション、Python、またはシステムの依存関係

#### 1.1.1. 自動化実行環境を使用する理由

自動化実行環境では、Red Hat Ansible Automation Platform はコントロールプレーンを実行プレーンから分離することで、分散アーキテクチャーに移行されました。コントロールプレーンから独立して自動化の実行を維持すると、開発サイクルが短縮され、環境間のスケーラビリティ、信頼性、および移植性が向上します。Red Hat Ansible Automation Platform には、Ansible コンテンツツールへのアクセスも含まれているため、自動化実行環境の構築と管理が容易になります。

自動化実行環境には、速度、移植性、柔軟性に加え、以下の利点があります。

- これにより、自動化が複数のプラットフォームで一貫して実行し、システムレベルの依存関係とコレクションベースのコンテンツを組み込むことができます。
- これにより、Red Hat Ansible Automation Platform の管理者は、さまざまなチームのニーズを満たす自動化環境を提供し、管理できるようになります。
- これにより、自動化環境を構築して配布する標準的な方法を提供することで、自動化を簡単に拡張およびチーム間で共有できます。
- これにより、自動化チームは自動化環境自体の定義、構築、および更新を行うことができます。
- 自動化実行環境は、自動化の依存関係を通信するための共通の言語を提供します。

## 第2章 ANSIBLE BUILDER の使用

Ansible Builder は、さまざまな Ansible Collections で定義されたメタデータを使用するか、ユーザーが作成したメタデータを使用して自動化実行環境を構築するプロセスを自動化するコマンドラインツールです。

### 2.1. ANSIBLE BUILDER を使用する理由

Ansible Builder が開発される前に、Red Hat Ansible Automation Platform ユーザーは、必要なすべての依存関係がインストールされているカスタムの仮想環境またはコンテナの作成時に、依存関係の問題およびエラーを実行することができました。

Ansible Builder を使用して、自動化実行環境に追加するコンテンツ (コレクション、要件、システムレベルパッケージなど) を指定するカスタマイズ可能な自動化実行環境定義ファイルを簡単に作成できるようになりました。これにより、ジョブの実行に必要な要件と依存関係をすべて満たすことができます。

### 2.2. ANSIBLE BUILDER のインストール

Red Hat Subscription Management (RHSM) を使用して Ansible Builder をインストールし、Red Hat Ansible Automation Platform サブスクリプションをアタッチできます。[Red Hat Ansible Automation Platform サブスクリプションをアタッチ](#)すると、**ansible-builder** のインストールに必要なサブスクリプションのみのリソースにアクセスできます。サブスクリプションをアタッチすると、**ansible-builder** に必要なリポジトリが自動的に有効になります。



#### 注記

**ansible-builder** をインストールする前に、有効なサブスクリプションがホストにアタッチされている必要があります。

#### 手順

- ターミナルで次のコマンドを実行して、Ansible Builder をインストールし、Ansible Automation Platform リポジトリをアクティベートします。

```
# dnf install --enablerepo ansible-automation-platform-2.2-for-rhel-8-x86_64-rpms ansible-builder
```

### 2.3. 定義ファイルの構築

Ansible Builder をインストールしたら、自動化実行環境イメージを作成するために Ansible Builder が使用する定義ファイルを作成できます。自動化実行環境イメージを構築する高レベルのプロセスは、Ansible Builder が定義ファイルを読み取って検証してから、**Containerfile** を作成し、最後に **Containerfile** を Podman に渡して、自動化実行環境イメージを作成します。作成される定義ファイルは **yaml** 形式であり、異なるセクションが含まれます。定義ファイルの内容の詳細は、[定義ファイルの内容の内訳](#) を参照してください。

以下は定義ファイルの例です。

#### 例2.1 定義ファイル

```
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 ビルド引数のデフォルト値の一覧表示
- ❷ 各種要件ファイルの場所の指定
- ❸ 追加のカスタムビルド手順用のコマンド

これらの定義ファイルパラメーターの詳細は、[定義ファイルの内容の内訳](#) を参照してください。

## 2.4. ビルドの実行およびコマンドの作成

### 前提条件

- 定義ファイルを作成している

### 手順

自動化実行環境イメージをビルドするには、以下を実行します。

```
$ ansible-builder build
```

Ansible Builder は、デフォルトでは **execution-environment.yml** という名前の定義ファイルを探しますが、**-f** フラグを使用して異なるファイルパスを引数として指定できます。

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

**definition-file-name** は、定義ファイルの名前を指定します。

## 2.5. 定義ファイルの内容の内訳

自動化実行環境コンテナイメージに含まれるコンテンツを指定するため、Ansible Builder で自動化実行環境を構築するには、定義ファイルが必要です。

次のセクションでは、定義ファイルの内容を説明します。

### 2.5.1. ビルド引数およびベースイメージ

定義ファイルの **build\_arg\_defaults** セクションは、キーが Ansible Builder への引数のデフォルト値を指定できるディクショナリーです。**build\_arg\_defaults** で使用できる値の一覧は、以下の表を参照してください。

値	説明
<b>ANSIBLE_GALAXY_CLI_COLLECTION_OPTS</b>	コレクションのインストールフェーズで、ユーザーは ansible-galaxy CLI に任意の引数を渡すことができます。たとえば、プレリリースコレクションのインストールを有効にする <code>-pre</code> フラグや、サーバーの SSL 証明書の検証を無効にする <code>-c</code> などです。
<b>EE_BASE_IMAGE</b>	<p>自動化実行環境の親イメージを指定し、既存のイメージに基づいて新しいイメージを構築できるようにします。これは通常、<code>ee-minimal</code> や <code>ee-supported</code> などのサポートされる実行環境ベースイメージですが、以前に作成した実行環境イメージをカスタマイズすることもできます。</p> <p>デフォルトのイメージは <b>registry.redhat.io/ansible-automation-platform-23/ee-minimal-rhel8:latest</b> です。</p>
<b>EE_BUILDER_IMAGE</b>	<p>Python の依存関係コレクションとコンパイルに使用される中間ビルダーイメージを指定します。対応する Python バージョンと <b>EE_BASE_IMAGE</b> が含まれ、ansible-builder がインストールされている必要があります。</p> <p>デフォルトのイメージは <b>registry.redhat.io/ansible-automation-platform-23/ansible-builder-rhel8:latest</b> です。</p>

**build\_arg\_defaults** で指定される値は **Containerfile** にハードコーディングされるため、**podman build** を手動で呼び出すとこの値が維持されます。



#### 注記

CLI **--build-arg** フラグで同じ変数が指定されている場合は、CLI 値の優先順位が高くなります。

### 2.5.2. Ansible 設定ファイルパス

**ansible\_config** ディレクティブを使用すると、**ansible.cfg** ファイルへのパスを指定して、ビルドの Collection インストールステージで、プライベートアカウントのトークンおよびその他の設定を自動化ハブサーバーに渡すことができます。設定ファイルパスは定義ファイルの場所を基準にし、生成されたコンテナビルドコンテキストにコピーされます。

**ansible.cfg** ファイルは以下の例のような形式にする必要があります。

#### 例2.2 ansible.cfg ファイル

```
[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
```

自動化ハブからコレクションをダウンロードする方法は、関連する Ansible ドキュメントページ を参照してください。

### 2.5.3. 依存関係

自動化実行環境イメージの問題を回避するには、Galaxy、Python、およびシステムのエントリーが有効な要件ファイルを参照していることを確認してください。

#### 2.5.3.1. Galaxy

**galaxy** エントリーは、**ansible-galaxy collection install -r ...** コマンドの有効な要件ファイルを参照します。

**requirements.yml** エントリーは、自動化実行環境定義のディレクトリーからの相対パス、または絶対パスである可能性があります。

**requirements.yml** ファイルの内容は以下のようになります。

#### 例2.3 Galaxy の requirements.yml ファイル

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

#### 2.5.3.2. Python

定義ファイルの **python** エントリーは、**pip install -r ...** コマンドの有効な要件ファイルを参照します。

**requirements.txt** エントリーは、Collection がすでに Python の依存関係としてリストされているように追加の Python 要件をインストールするファイルです。自動化実行環境定義のフォルダーのディレクトリーからの相対パス、または絶対パスとして記載されている可能性があります。**requirements.txt** ファイルの内容は、**pip freeze** コマンドの標準出力と同様に、以下の例のような形式にする必要があります。

#### 例2.4 Python の requirements.txt ファイル

```
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. システム

定義の **システム** エントリーは、**bindep** 要件ファイルを指定しています。このファイルは、コレクションに依存関係として含まれていないシステムレベルの依存関係をインストールします。自動化実行環境定義のフォルダーのディレクトリーからの相対パス、または絶対パスとして記載されている可能性があります。最低限、コレクションが、**[platform:rpm]** に必要な要件を指定することが想定されます。

これは、**libxml2** パッケージおよび **subversion** パッケージをコンテナに追加する **bindep.txt** ファイルの例です。

#### 例2.5 bindep.txt ファイル

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

複数のコレクションからのエントリーは、1つのファイルに統合されます。これは **bindep** により処理され、**dnf** に渡されます。プロファイルのない要件や、ランタイム要件がイメージにインストールされません。

### 2.5.4. 追加のカスタムビルドの手順

**prepend** コマンドと **append** コマンドは、**additional\_build\_steps** セクションで指定できます。これにより、メインのビルド手順が実行される前または後に実行する **Containerfile** にコマンドが追加されます。

**additional\_build\_steps** の構文は以下のいずれかである必要があります。

- 複数行の文字列

#### 例2.6 複数行の文字列エントリー

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

- リスト

#### 例2.7 リストエントリー

```

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

```

## 2.6. 任意のビルドコマンド引数

**-t** フラグは、自動化実行環境イメージに特定の名前でタグ付けします。たとえば、以下のコマンドは **my\_first\_ee\_image** という名前のイメージを構築します。

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



### 注記

**build** で **-t** を使用しない場合は、**ansible-execution-env** というイメージが作成され、ローカルのコンテナレジストリーに読み込まれます。

複数の定義ファイルがある場合は、**-f** フラグで、使用する定義ファイルを指定できます。

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

[上記の例](#) では、Ansible Builder が、デフォルトの **execution-environment.yml** の代わりに **another-definition-file.yml** ファイルで提供される仕様を使用して、**another\_ee\_image** という名前の自動実行環境イメージを構築します。

**build** コマンドで利用できるその他の仕様とフラグについては、**ansible-builder build --help** と入力して、追加オプションのリストを表示してください。

## 2.7. CONTAINERFILE

定義ファイルが作成されると、Ansible Builder はこのファイルを読み取り、検証し、**Containerfile** を作成し、最後に **Containerfile** を Podman に渡してパッケージ化し、以下の手順を使用して自動化実行環境イメージを作成します。

1. ベースイメージを取得します。
2. ベースイメージの一時コピーでは、コレクションがダウンロードされ、宣言された Python およびシステム依存関係の一覧 (存在する場合) が後に収集されます。
3. エフェメラルビルダーイメージでは、定義ファイルに記載されているすべての Python 依存関係の Python ホイールがダウンロードされ、(必要に応じて) ビルドされます。これには、定義ファイルに記載されているコレクションによって宣言されたすべての Python 依存関係が含まれます。
4. 定義ファイルの **additional\_build\_steps** に対して **prepend** が実行します。
5. 最終的な自動化実行環境イメージでは、定義ファイルに記載されているシステム依存関係がインストールされ、定義ファイルに記載されているコレクションが宣言したすべてのシステム依存関係が含まれます。
6. 最終的な自動化実行環境イメージでは、ダウンロードしたコレクションがコピーされ、以前にフェッチされた Python 依存関係がインストールされます。

7. 定義ファイルの `additional_build_steps` に対して **append** が実行します。

## 2.8. イメージの構築なしで **CONTAINERFILE** の作成

イメージをビルドせずに共有可能な **Containerfile** を作成するには、以下を実行します。

```
$ ansible-builder create
```

## 第3章 自動化実行環境の公開

### 3.1. 既存の自動化実行環境イメージのカスタマイズ

Ansible コントローラーには、以下の 3 つのデフォルト実行環境が同梱されています。

- **Ansible 2.9** - Controller モジュール以外のコレクションがインストールされない
- **Minimal** - Ansible Runner とともに最新の Ansible 2.13 リリースが含まれるが、コレクションや他の追加コンテンツは含まれない
- **EE Supported** - 最小限のもの、および Red Hat がサポートするすべてのコレクションと依存関係

これらの環境は多くの自動化ユースケースに対応しますが、追加の項目を追加して、特定のニーズに合わせてこれらのコンテナをカスタマイズできます。以下の手順では、**kubernetes.core** コレクションを **ee-minimal** デフォルトイメージに追加します。

#### 手順

1. Podman を使用して **registry.redhat.io** にログインします。

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

2. 必要な自動化実行環境のベースイメージをプルできることを確認します。

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

3. 必要なベースイメージと新しい実行環境イメージに追加する追加のコンテンツを指定するように Ansible Builder ファイルを設定します。

- a. たとえば、[Kubernetes Core Collection](#) を [Galaxy から](#) イメージに追加するには、以下のよう  
に **requirements.yml** ファイルを入力します。

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

- b. 定義ファイルとそのコンテンツの詳細は、[定義ファイルの内訳](#) セクションを参照してください。

4. 実行環境定義ファイルで、**EE\_BASE\_IMAGE** フィールドに元の **ee-minimal** コンテナの URL とタグを指定します。その場合、最終的な **execution-environment.yml** ファイルは以下  
のようになります。

#### 例3.1 カスタマイズされた **execution-environment.yml** ファイル

```
version: 1

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

dependencies:
  galaxy: requirements.yml
```



### 注記

この例では、自動化ハブの認定コレクションではなく、コミュニティバージョンの **kubernetes.core** を使用するため、**ansible.cfg** ファイルを作成したり、定義ファイルで参照したりする必要はありません。

5. 以下のコマンドを使用して、新しい実行環境イメージを構築します。

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

ここで、**[username]** はユーザー名を指定し、**new-ee** は新しいコンテナイメージの名前を指定します。



### 注記

**build** で **-t** を使用しない場合は、**ansible-execution-env** というイメージが作成され、ローカルのコンテナレジストリーに読み込まれます。

- a. **podman images** コマンドを使用して、新しいコンテナイメージが一覧に表示されていることを確認します。

#### 例3.2 new-ee イメージを使用した podman images コマンドの出力

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

1. コレクションがインストールされていることを確認します。

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

2. Automation Hub で使用するイメージにタグを付けます。

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

3. Podman を使用して Automation Hub にログインします。



### 注記

コンテナをプッシュするには、Automation Hub の **admin** または適切なコンテナリポジトリのアクセス許可が必要です。詳細は、[Red Hat Ansible Automation Platform ドキュメントのプライベート Automation Hub でのコンテナの管理](#) を参照してください。

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

4. イメージを自動化ハブのコンテナレジストリーにプッシュします。

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

5. 新しいイメージを自動化コントローラーインスタンスにプルします。
- b. Automation Controller に移動します。
- c. ナビゲーションバーから、**Administration** → **Execution Environments** の順にクリックします。
- d. **Add** をクリックします。
- e. 適切な情報を入力して **Save** をクリックし、新規イメージにプルします。



#### 注記

自動化ハブのインスタンスがパスワードまたはトークンで保護されている場合は、適切なコンテナーレジストリーの認証情報が設定されていることを確認してください。

## 第4章 プライベート自動化ハブコンテナーレジストリーへの入力

デフォルトでは、プライベート自動化ハブにはコンテナイメージが含まれません。コンテナレジストリーを設定するには、コンテナイメージレジストリーにコンテナイメージをプッシュする必要があります。本セクションの手順では、Red Hat Ecosystem Catalog (registry.redhat.io) からイメージをプルし、それらをタグ付けして、プライベート自動化コンテナレジストリーにプッシュする方法を説明します。

### 4.1. 前提条件

- 新しいコンテナを作成して、プライベート自動化ハブにプッシュするパーミッションがある。

### 4.2. 自動化ハブで使用するイメージの取得

コンテナイメージをプライベート自動化ハブにプッシュする前に、まず既存のレジストリーからプルし、使用できるようにタグを付ける必要があります。以下の例では、Red Hat Ecosystem Catalog (registry.redhat.io) からイメージをプルする方法を説明します。

#### 前提条件

registry.redhat.io からイメージをプルするパーミッションがある。

#### 手順

1. registry.redhat.io 認証情報を使用して Podman にログインします。

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

2. プロンプトにユーザー名およびパスワードを入力します。
3. コンテナイメージをプルします。

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

#### 検証

1. ローカルストレージ内のイメージを一覧表示します。

```
$ podman images
```

2. 最近プルしたイメージが一覧に含まれていることを確認します。
3. タグが正しいことを確認します。

#### 関連情報

- イメージの登録および取得についての詳細は、[Red Hat Ecosystem Catalog Help](#) を参照してください。

### 4.3. 自動化ハブで使用するイメージのタグ付け

レジストリーからイメージをプルしたら、プライベート自動化ハブコンテナレジストリーで使用するようにタグを付けます。

#### 前提条件

- 外部レジストリーからコンテナイメージをプルしている。
- 自動化ハブインスタンスの FQDN または IP アドレスがある。

#### 手順

- 自動化ハブコンテナレジストリーを使用してローカルイメージにタグを付けます。

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

#### 検証

1. ローカルストレージ内のイメージを一覧表示します。

```
$ podman images
```

2. 自動化ハブの情報で最近タグ付けされたイメージが一覧に含まれていることを確認します。

## 4.4. プライベート自動化ハブへのコンテナイメージのプッシュ

タグ付けされたコンテナイメージをプライベート自動化ハブにプッシュして新規コンテナを作成し、コンテナレジストリーにデータを入力できます。

#### 前提条件

- 新規コンテナを作成するパーミッションがある。
- 自動化ハブインスタンスの FQDN または IP アドレスがある。

#### 手順

1. 自動化ハブの場所および認証情報を使用して Podman にログインします。

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

2. コンテナイメージを自動化ハブのコンテナレジストリーにプッシュします。

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



## 注記

registry.redhat.io からの署名付きイメージが自動化ハブコンテナーレジストリーにプッシュされる場合は、**--remove-signatures** フラグが必要です。**push** 操作は、アップロード中にイメージレイヤーを再圧縮します。これは、再現性が保証されておらず、クライアントの実装に依存します。これにより、イメージレイヤーダイジェストが変更され、プッシュ操作が失敗し、**Error: Copying this image requires changing layer representation, which is not possible (image is signed or the destination specifies a digest)** エラーが発生します。

## 検証

1. ローカルの自動化ハブにログインします。
2. **Container Registry** に移動します。
3. コンテナーリポジトリ一覧でコンテナーを見つけます。

## 第5章 コンテナリポジトリの設定

コンテナリポジトリを設定して説明の追加、README の追加、リポジトリにアクセスできるグループの追加、およびタグイメージの追加を行うことができます。

### 5.1. 前提条件

- リポジトリを変更するパーミッションを持つプライベート Automation Hub にログインしている。

### 5.2. コンテナリポジトリへの README の追加

コンテナリポジトリに README を追加して、コンテナを操作する方法をユーザーに提供します。自動化ハブコンテナリポジトリは、README を作成するためのマークダウンをサポートします。デフォルトでは、README は空になります。

#### 前提条件

- コンテナを変更するパーミッションがある。

#### 手順

1. **Execution Environments** に移動します。
2. コンテナリポジトリを選択します。
3. **Detail** タブで、**Add** をクリックします。
4. **Raw Markdown** テキストフィールドに、Markdown で README テキストを入力します。
5. 終了したら **Save** をクリックします。

README を追加したら、**Edit** をクリックし、ステップ 4 および 5 を繰り返すことで、いつでも編集できます。

### 5.3. コンテナリポジトリへのアクセスの提供

イメージを使用する必要があるユーザーにコンテナリポジトリへのアクセスを提供します。グループを追加すると、グループがコンテナリポジトリに対して持つことができるパーミッションを変更できます。このオプションを使用して、グループが割り当てられている内容に応じてパーミッションを拡張または制限できます。

#### 前提条件

- コンテナの名前空間のパーミッションを変更している。

#### 手順

1. **Execution Environments** に移動します。
2. コンテナリポジトリを選択します。
3. ウィンドウの右上にある **Edit** をクリックします。

4. **Groups with access** で、アクセス権を付与するグループ (単数または複数) を選択します。

- (必要に応じて) グループ名でドロップダウンを使用して、特定のグループのパーミッションを追加または削除します。

5. **Save** をクリックします。

## 5.4. コンテナイメージのタグ付け

自動化ハブコンテナリポジトリに保存されているイメージにタグを付けて、名前を追加します。イメージにタグが追加されない場合、自動化ハブの名前はデフォルトで **latest** に設定されます。

### 前提条件

- **change image tags** のパーミッションがある。

### 手順

1. **Execution Environments** に移動します。

2. コンテナリポジトリを選択します。

3. **Images** タブをクリックします。

4.  をクリックし、**Manage tags** をクリックします。

5. テキストフィールドに新しいタグを追加し、**Add** をクリックします。

- (必要に応じて) そのイメージのいずれのタグの **x** をクリックして、**current tags** を削除します。

6. **Save** をクリックします。

### 検証

1. **Activity** タブをクリックし、最新の変更を確認します。

## 第6章 コンテナリポジトリからのイメージのプル

自動化ハブコンテナレジストリーからイメージを取得し、ローカルマシンにコピーを作成します。自動化ハブは、コンテナリポジトリの **latest** イメージごとに、**podman pull** コマンドを提供します。このコマンドを端末にコピーアンドペーストするか、**podman pull** を使用してイメージタグに基づいてイメージをコピーすることができます。

### 6.1. 前提条件

- プライベートコンテナリポジトリを表示し、プルするパーミッションがある。

### 6.2. イメージのプル

自動化ハブコンテナレジストリーからイメージをプルして、ローカルマシンにコピーできます。自動化ハブは、コンテナリポジトリの **latest** イメージごとに、**podman pull** コマンドを提供します。

#### 手順

1. **Execution Environments** に移動します。
2. コンテナリポジトリを選択します。
3. **Pull this image** エントリーで、**Copy to clipboard** をクリックします。
4. 端末でコマンドを貼り付けます。

#### 検証

1. **podman images** を実行して、ローカルマシンにイメージを表示します。

### 6.3. 関連情報

- イメージをプルする際に使用するオプションについては、[Podman ドキュメント](#) を参照してください。

## 付録A 自動化実行環境の優先順位

プロジェクト更新は常にコントロールプレーンの自動化実行環境を使用しますが、ジョブは以下のように最初に利用可能な自動化実行環境を使用します。

1. ジョブを作成したテンプレート (ジョブテンプレートまたはインベントリースource) で定義される **execution\_environment**。
2. ジョブが使用するプロジェクトで定義される **default\_environment**。
3. ジョブの組織で定義される **default\_environment**。
4. ジョブが使用するインベントリーの組織で定義される **default\_environment**。
5. 現在の **DEFAULT\_EXECUTION\_ENVIRONMENT** 設定 ([api/v2/settings/jobs/](#) で設定可能)。
6. **GLOBAL\_JOB\_EXECUTION\_ENVIRONMENTS** 設定からのイメージ。
7. その他のグローバル実行環境。



### 注記

複数の実行環境が基準 (6 および 7 に適用) に適合する場合は、最近作成された実行環境が使用されます。