



Red Hat Ansible Automation Platform 2.4

実行環境の作成および消費

Ansible Builder で実行環境を作成および使用する

Red Hat Ansible Automation Platform 2.4 実行環境の作成および消費

Ansible Builder で実行環境を作成および使用する

法律上の通知

Copyright © 2024 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

概要

このガイドでは、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する方法を説明します。

目次

はじめに	3
RED HAT ドキュメントへのフィードバック (英語のみ)	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. 自動化実行環境イメージのビルド	8
2.5. 定義ファイルの内容の内訳	8
2.6. 任意のビルドコマンド引数	12
2.7. CONTAINERFILE	13
2.8. イメージをビルドせずに CONTAINERFILE を作成する	13
第3章 一般的な自動化実行環境のシナリオ	14
3.1. AUTOMATION HUB CA 証明書の更新	14
3.2. 自動化実行環境を構築する際に AUTOMATION HUB 認証の詳細を使用する	14
3.3. 関連情報	15
第4章 自動化実行環境の公開	16
4.1. 既存の自動化実行環境イメージのカスタマイズ	16
4.2. 関連情報 (または次の手順)	18
第5章 PRIVATE AUTOMATION HUB コンテナレジストリーへの入力	19
5.1. AUTOMATION HUB で使用するイメージのプル	19
5.2. AUTOMATION HUB で使用するイメージのタグ付け	21
5.3. PRIVATE AUTOMATION HUB へのコンテナイメージのプッシュ	21
第6章 コンテナレジストリーの設定	23
6.1. コンテナレジストリーを設定するための前提条件	23
6.2. コンテナレジストリーへの README の追加	23
6.3. コンテナレジストリーへのアクセスの提供	23
6.4. コンテナイメージのタグ付け	24
6.5. AUTOMATION CONTROLLER での認証情報の作成	24
第7章 コンテナレジストリーからのイメージのプル	26
7.1. イメージのプル	26
7.2. コンテナレジストリーからのイメージの同期	26
付録A 自動化実行環境の優先順位	28

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。

RED HAT ドキュメントへのフィードバック (英語のみ)

このドキュメントの改善に関するご意見がある場合や、エラーを発見した場合は、<https://access.redhat.com> から Technical Support チームに連絡してください。

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

Red Hat Ansible Automation Platform のすべての自動化は、自動化実行環境と呼ばれるコンテナイメージ上で実行されます。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、以下が含まれている必要があります。

- Ansible Core 2.15 以降
- Python 3.8-3.11
- Ansible Runner
- Ansible コンテンツコレクションとその依存関係
- 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 を使用すると、Ansible Core、Python、Collection、サードパーティーの Python 要件、システムレベルのパッケージなどの自動化実行環境に含むコンテンツを指定する、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

前提条件

- Podman コンテナランタイムがインストールされている。
- ホストに有効なサブスクリプションがアタッチされている。これにより、**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、**ansible-builder** に必要なリポジトリが自動的に有効化されるようになります。詳細は、[Red Hat Ansible Automation Platform サブスクリプションのアタッチ](#) を参照してください。

手順

1. ターミナルで次のコマンドを実行して、Ansible Automation Platform リポジトリをアクティブ化します。

```
# dnf install --enablerepo=ansible-automation-platform-2.4-for-rhel-9-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後、Ansible Builder が自動化実行環境イメージを作成するために使用する定義ファイルを作成できます。Ansible Builder は、定義ファイルを読み取って検証してから **Containerfile** を作成し、最後にその **Containerfile** を Podman に渡して自動化実行環境イメージを作成します。続いて Podman は、自動化実行環境イメージをパッケージ化して作成します。作成する定義ファイルは **yaml** 形式であり、さまざまなセクションが含まれている必要があります。指定されていない場合のデフォルトの定義ファイル名は、**execution-environment.yml** です。定義ファイルの各部分の詳細は、[定義ファイルの内容の内訳](#) を参照してください。

以下はバージョン 3 の定義ファイルの例です。各定義ファイルでは、使用する Ansible Builder 機能セットのメジャーバージョン番号を指定する必要があります。指定しない場合、Ansible Builder はデフォルトでバージョン 1 になり、ほとんどの新機能と定義キーワードが使用できなくなります。

例2.1 定義ファイルの例

```
version: 3
```

```
build_arg_defaults: ❶
  ANSIBLE_GALAXY_CLI_COLLECTION_OPTS: '--pre'

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

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

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

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

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

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

- ❶ ビルド引数のデフォルト値をリストします。
- ❷ 各種要件ファイルの場所を指定します。
- ❸ 使用するベースイメージを指定します。Red Hat のサポートは、redhat.registry.io のベースイメージに対してのみ提供されます。
- ❹ Builder ランタイム機能に影響を与える可能性があるオプションを指定します。
- ❺ 追加のカスタムビルド手順用のコマンド。

関連情報

- 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。
- Ansible Builder バージョン 2 と 3 の違いの詳細は、[Ansible 3 Porting Guide](#) を参照してください。

2.4. 自動化実行環境イメージのビルド

定義ファイルを作成したら、自動化実行環境イメージのビルドに進むことができます。

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、コマンドラインから次のコマンドを実行します。

```
$ 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
ANSIBLE_GALAXY_CLI_COLLECTION_OPTS	コレクションのインストールフェーズで、ユーザーは <code>ansible-galaxy CLI</code> 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。
ANSIBLE_GALAXY_CLI_ROLE_OPTS	使用すると、 <code>-no-deps</code> などのフラグをロールのインストールに渡すことができます。

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。

最終イメージにインストールする必要がある依存関係は、定義ファイルの依存関係セクションに含めることができます。

自動化実行環境イメージに関する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指していること、またはそれぞれのファイルタイプに対して有効なコンテンツであることを確認してください。

2.5.1.1. Galaxy

galaxy エントリーには、有効な要件ファイルへの参照か、**ansible-galaxy collection install -r ...** コマンドのインラインコンテンツを含めます。

requirements.yml エントリーは、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か、絶対パスにすることができます。

内容は次のようになります。

例2.2 Galaxy エントリー

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

2.5.1.2. Python

定義ファイル内の **Python** エントリーは、有効な要件ファイル、または **pip install -r ...** コマンドの PEP508 形式の Python 要件のインラインリストを参照します。

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2.3 Python エントリー

```
boto>=2.49.0
botocore>=1.12.249
pytz
python-dateutil>=2.7.0
awxkit
packaging
requests>=2.4.2
xmldict
azure-cli-core==2.11.1
```

```
openshift>=0.6.2
requests-oauthlib
openstacksdk>=0.13
ovirt-engine-sdk-python>=4.4.10
```

2.5.1.3. システム

定義内の **system** エントリは、**bindep** 要件ファイルまたは **bindep** エントリのインラインリストを指します。これらは、コレクションに依存関係としてすでに含まれているものの範囲外にある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リスト表示できます。少なくとも、コレクションでは **[platform:rpm]** に必要な要件を指定する必要があります。

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2.4 System エントリー

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

複数のコレクションからのエントリは、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。

2.5.2. イメージ

定義ファイルの **イメージ** セクションでは、ベースイメージを識別します。署名付きコンテナイメージの検証は、**podman** コンテナランタイムでサポートされています。

イメージで使用できる値のリストについては、次の表を参照してください。

値	説明
base_image	<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ビルドできるようにします。これは通常、ee-minimal または ee-supported などのサポートされる実行環境ベースイメージですが、作成済みでさらにカスタマイズしたい実行環境イメージでも問題ありません。</p> <p>コンテナイメージが使用するには name キーが必要です。イメージがリポジトリ内でミラーリングされているが、イメージの元の署名キーで署名されている場合は、signature_original_name キーを指定します。イメージ名には、:latest などのタグが含まれている必要があります。</p> <p>デフォルトのイメージは registry.redhat.io/ansible-automation-platform-24/ee-minimal-rhel8:latest です。</p>



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。

2.5.3. 追加のビルドファイル

定義ファイルの `additional_build_steps` セクションに外部ファイルを参照またはコピーすることで、任意の外部ファイルをビルドコンテキストディレクトリーに追加できます。形式はディクショナリー値のリストで、各リスト項目に `src` および `dest` のキーと値を含めます。

各リスト項目は、次の必須キーを含むディクショナリーである必要があります。

src

ビルドコンテキストディレクトリーにコピーするソースファイルを指定します。これは、絶対パス (`/home/user/.ansible.cfg` など) か、実行環境ファイルへの相対パスにすることができます。相対パスは、1つ以上のファイルに一致する glob 式にすることができます (`files/*.cfg` など)。



注記

絶対パスには正規表現を含めることはできません。`src` がディレクトリーの場合、そのディレクトリーの内容全体が `dest` にコピーされます。

dest

ソースファイル (例: `files/configs`) が含まれるビルドコンテキストディレクトリーの `_build` サブディレクトリーの下にあるサブディレクトリーパスを指定します。この値を絶対パスにしたり、パス内に `..` を含めたりすることはできません。このディレクトリーが存在しない場合は、Ansible Builder によって作成されます。

2.5.4. 追加のカスタムビルドの手順

定義ファイルの `additional_build_steps` セクションで、任意のビルドフェーズのカスタムビルドコマンドを指定できます。これにより、ビルドフェーズをきめ細かく制御できるようになります。

`prepend_` および `append_` コマンドを使用して、メインのビルドステップの実行前または実行後に実行されるディレクティブを `Containerfile` に追加します。コマンドは、ランタイムシステムに必要なルールに準拠する必要があります。

`added_build_steps` で使用できる値のリストは、次の表を参照してください。

値	説明
<code>prepend_base</code>	ベースイメージをビルドする前にコマンドを挿入できます。
<code>append_base</code>	ベースイメージをビルドした後にコマンドを挿入できます。
<code>prepend_galaxy</code>	galaxy イメージをビルドする前に挿入できます。
<code>append_galaxy</code>	galaxy イメージをビルドした後に挿入できます。

値	説明
<code>prepend_builder</code>	Python ビルダーイメージをビルドする前に、コマンドを挿入できます。
<code>append_builder</code>	Python ビルダーイメージをビルドした後に、コマンドを挿入できます。
<code>prepend_final</code>	最終イメージをビルドする前に挿入できます。
<code>append_final</code>	最終イメージをビルドした後に挿入できます。

`added_build_steps` の構文は、複数行の文字列とリストの両方をサポートします。以下の例を参照してください。

例2.5 複数行の文字列エントリー

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

例2.6 リストエントリー

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

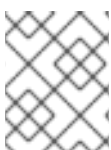
2.5.5. 関連情報

- 一般的なシナリオの定義ファイルの例については、[Ansible Builder ドキュメントの COMMON SCENARIOS](#) セクションを参照してください。

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称でタグ付けします。たとえば、以下のコマンドは `my_first_ee_image` という名称のイメージをビルドします。

```
$ 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` フラグを含めることで、どれを使用するかを指定できます。


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

この例 では、Ansible Builder は、デフォルトの **execution-environment.yml** の代わりに、**another-definition-file.yml** という名前の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ビルドします。

build コマンドで使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ます。

2.7. CONTAINERFILE

定義ファイルが作成されると、Ansible Builder はそれを読み取って検証し、**Containerfile** とコンテナビルドコンテキストを作成し、オプションでこれらを Podman に渡して自動化実行環境イメージをビルドします。コンテナのビルドは、**base**、**galaxy**、**builder**、および **final** といういくつかの異なるステージで行われます。イメージのビルド手順 (および、**additional_build_steps** で定義された対応するカスタムの **prepend_** および **append_** 手順) は次のとおりです。

1. **base** ビルドステージでは、指定されたベースイメージが、Python、**pip**、**ansible-core**、**ansible-runner** など、他のビルドステージに必要なコンポーネントを使用して (オプションで) カスタマイズされます。次に、生成されたイメージが検証され、必要なコンポーネントが利用可能であることが確認されます (コンポーネントはベースイメージにすでに存在している可能性があるため)。結果として得られるカスタマイズされた **base** イメージの一時的なコピーは、他のすべてのビルドステージのベースとして使用されます。
2. **galaxy** ビルドステージでは、定義ファイルで指定されたコレクションがダウンロードされ、**final** ビルドステージでのインストールのために保存されます。コレクションによって宣言された Python とシステムの依存関係も (存在する場合)、今後の分析のために収集されます。
3. **builder** ビルドステージでは、コレクションによって宣言された Python の依存関係が、定義ファイルにリストされている依存関係とマージされます。この最後の Python 依存関係セットは、Python ホイールとしてダウンロードおよびビルドされ、**final** ビルドステージでのインストールのために保存されます。
4. **final** ビルドステージでは、以前にダウンロードしたコレクションが、システムパッケージおよびコレクションによって依存関係として宣言された、または定義ファイルにリストされていた、以前にビルドされた Python パッケージとともにインストールされます。

2.8. イメージをビルドせずに CONTAINERFILE を作成する

セキュリティ上の理由から、サンドボックス環境でビルドした共有コンテナイメージを使用する必要がある場合は、共有可能な **Containerfile** を作成できます。

```
$ ansible-builder create
```

第3章 一般的な自動化実行環境のシナリオ

一般的な設定シナリオに対応するには、次の定義ファイルの例を使用します。

3.1. AUTOMATION HUB CA 証明書の更新

この例を使用して、デフォルトの定義ファイルをカスタマイズして CA 証明書を **additional-build-files** セクションに含め、そのファイルを適切なディレクトリーに移動します。最後にコマンドを実行して CA 証明書の動的設定を更新し、システムがこの CA 証明書を信頼できるようにします。

前提条件

- カスタム CA 証明書 (例: **rootCA.crt**)。



注記

prepend_base を使用して CA 証明書をカスタマイズすると、その結果作成される CA 設定が、他のすべてのビルドステージと最終イメージに表示されます。他のすべてのビルドステージは、ベースイメージから継承されるためです。

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

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

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

[galaxy]
server_list = automation_hub
```

3.2. 自動化実行環境を構築する際に AUTOMATION HUB 認証の詳細を使用する

以下の例を使用して、デフォルトの定義ファイルをカスタマイズし、Automation Hub 認証の詳細を、最終的な自動化実行環境イメージで公開することなく、自動化実行環境ビルドに渡します。

前提条件

- [Automation Hub API トークンを作成](#) し、セキュアな場所 (たとえば **token.txt** という名前のファイル) に保存している。
- Automation Hub API トークンが入力されるビルド引数を定義している。

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

```
additional_build_steps:
```

```
  prepend_galaxy:
```

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

```
    # `--build-arg ANSIBLE_GALAXY_SERVER_AUTOMATION_HUB_TOKEN` to get it to pick it up
```

```
from the host env
```

```
  - ARG ANSIBLE_GALAXY_SERVER_AUTOMATION_HUB_TOKEN
```

```
  - ENV ANSIBLE_GALAXY_SERVER_LIST=automation_hub
```

```
  - ENV
```

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

```
  - ENV
```

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

3.3. 関連情報

- 自動化実行環境の定義ファイルの各部分については、[定義ファイルの内容の内訳](#)を参照してください。
- その他の一般的なシナリオ用の定義ファイルの例は、[Ansible Builder Documentation](#)の[一般的なシナリオのセクション](#)を参照してください。

第4章 自動化実行環境の公開

4.1. 既存の自動化実行環境イメージのカスタマイズ

Ansible Controller には、以下のデフォルトの実行環境が含まれます。

- **最小限** - 最新の Ansible-core 2.15 リリースと Ansible Runner が含まれますが、コレクションやその他のコンテンツは含まれません。
- **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-24/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、[Galaxy の Kubernetes Core Collection](#) をイメージに追加するには、Galaxy エントリーを使用します。

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

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

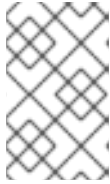
4. 実行環境定義ファイルで、**EE_BASE_IMAGE** フィールドに元の **ee-minimal** コンテナの URL とタグを指定します。その場合、最終的な **execution-environment.yml** ファイルは以下ようになります。

例4.1 カスタマイズされた execution-environment.yml ファイル

```
version: 3

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

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



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。

- 以下のコマンドを使用して、新しい実行環境イメージをビルドします。

```
$ ansible-builder build -t [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。

- **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4.2 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

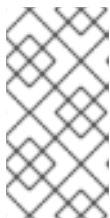
- コレクションがインストールされていることを確認します。

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

- Automation Hub で使用するイメージにタグを付けます。

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

- 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、[Automation Hub でのコンテンツの管理](#) の **Private Automation Hub でのコンテナの管理** セクションを参照してください。

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

- 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
```

- 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。

- Automation Controller に移動します。

- a. Automation Controller に移動します。
- b. ナビゲーションパネルから、**Administration** → **Execution Environments** を選択します。
- c. **Add** をクリックします。
- d. 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。

4.2. 関連情報 (または次の手順)

一般的なシナリオに基づいて実行環境をカスタマイズする方法の詳細は、**Ansible Builder ドキュメント**の次のトピックを参照してください。

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

第5章 PRIVATE AUTOMATION HUB コンテナレジストリーへの入力

デフォルトでは、Private Automation Hub にはコンテナイメージが含まれていません。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。

Private Automation Hub コンテナレジストリーを設定するには、特定のワークフローに従う必要があります。

- Red Hat Ecosystem Catalog (registry.redhat.io) からイメージをプルします。
- イメージにタグを付けます。
- Private Automation Hub コンテナレジストリーにプッシュします。

重要

イメージマニフェストとファイルシステム Blob はどちらも、元々は registry.redhat.io および registry.access.redhat.com から直接提供されていました。2023 年 5 月 1 日以降、ファイルシステム Blob は代わりに quay.io から提供されます。

- [ネットワークポートとプロトコル](#) (表 5.10. 実行環境 (EE) が、コンテナイメージをプルする際の問題を回避するのに使用できることを確認してください。

registry.redhat.io または registry.access.redhat.com への送信接続を特に有効にするすべてのファイアウォール設定にこの変更を加えます。

ファイアウォールルールを設定するときは、IP アドレスの代わりにホスト名を使用します。

この変更を行った後、引き続き registry.redhat.io および registry.access.redhat.com からイメージをプルできます。Red Hat コンテナイメージのプルを続行するためには、quay.io にログインする必要も、quay.io レジストリーと直接やりとりする必要もありません。

ただし、Web ベースの Red Hat Subscription Management 上のマニフェスト (「サブスクリプション割り当て」とも呼ばれる) は、1つの例外を除いて、2024 年初頭にサポートされなくなりました。システムが、Red Hat のサーバーから直接更新を受信しないクラウドネットワークまたは「エアギャップ」システムの一部である場合、マニフェストは Red Hat Satellite 6.16 のリリースまでサポートされます。Red Hat Satellite 6.16 のリリース日の発表については、[Red Hat Satellite のリリース日](#) を確認してください。

5.1. AUTOMATION HUB で使用するイメージのプル

コンテナイメージを Private Automation Hub 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。次の例では、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。



重要

2024 年初頭の時点で、Red Hat は Red Hat Subscription Management Web プラットフォーム上のマニフェストまたはマニフェストリスト (「サブスクリプション割り当て」と同義で使用) をサポートしなくなりました。Red Hat は、1つの例外を除いて、Red Hat Satellite のマニフェスト機能のほとんどをサポートしなくなりました: * Red Hat サーバーから直接更新を受信しないクローズドネットワークまたは「エアギャップ」ネットワーク内の Red Hat Satellite ユーザーは、Red Hat Satellite 6.16 がリリースされるまで **access.redhat.com** を引き続き使用できます。

新しい Red Hat アカウントでは、サブスクリプションツールに Simple Content Access が自動的に使用されます。新しい Red Hat アカウントと、Red Hat のサーバーに接続できる既存の Satellite のお客様は、**console.redhat.com** でマニフェストを見つけることができます。

前提条件

- registry.redhat.io からイメージをプルする権限がある。
- Simple Content Access が有効になっている Red Hat アカウント。

手順

1. コンテナイメージのマニフェストにアクセスする必要がある場合は、[Red Hat コンソール](#) にログインします。
2. コンテナイメージに必要なマニフェストの3つのドットメニューをクリックし、**Export manifest** をクリックします。
3. registry.redhat.io の認証情報を使用して Podman にログインします。

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

4. ユーザー名およびパスワードを入力します。
5. 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

最近プルしたイメージがリストに含まれていることを確認するには、次の手順を実行します。

1. ローカルストレージ内のイメージをリスト表示します。

```
$ podman images
```

2. イメージ名を確認し、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関する詳細は、[Red Hat Ecosystem Catalog Help](#) を参照してください。

- Red Hat サブスクリプションツールの変更点に関する詳細は、[接続された Satellite Server のマニフェストの作成と管理](#) を参照してください。

5.2. AUTOMATION HUB 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、Private Automation Hub 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。
- Automation Hub インスタンスの FQDN または IP アドレスがある。

手順

- Automation Hub 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_hostname>/<container_image_name>
```

検証

1. ローカルストレージ内のイメージをリスト表示します。

```
$ podman images
```

2. Automation Hub 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。

5.3. PRIVATE AUTOMATION HUB 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 Private Automation Hub にプッシュして、新しいコンテナを作成し、コンテナレジストリーに追加できます。

前提条件

- 新規コンテナを作成する権限がある。
- Automation Hub インスタンスの FQDN または IP アドレスがある。

手順

1. Automation Hub 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 Automation Hub 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>
```

トラブルシューティング

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。この操作は、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. Automation Hub にログインします。
2. **Container Registry** に移動します。
3. コンテナリポジトリリストでコンテナを見つけます。

第6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するには、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、およびイメージのタグ付けを行う必要があります。

6.1. コンテナレジストリーを設定するための前提条件

- Private Automation Hub にログインしている。
- リポジトリを変更する権限が必要です。

6.2. 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。Automation Hub 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 は空です。

前提条件

- コンテナを変更する権限がある。

手順

1. Automation Hub にログインします。
2. ナビゲーションパネルから、**Execution Environments** → **Execution Environments** を選択します。
3. コンテナリポジトリを選択します。
4. **Detail** タブで、**Add** をクリックします。
5. **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。
6. 完了したら、**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。

6.3. コンテナリポジトリへのアクセスの提供

イメージを操作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権限を変更できます。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権限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権限を変更している。

手順

1. Automation Hub にログインします。

2. ナビゲーションパネルから、**Execution Environments** → **Execution Environments** を選択します。
3. コンテナリポジトリを選択します。
4. **Access** タブで、**Select a group** をクリックします。
5. アクセスを許可するグループを選択し、**Next** をクリックします。
6. この実行環境に追加するロールを選択し、**Next** をクリックします。
7. **Add** をクリックします。

6.4. コンテナイメージのタグ付け

Automation Hub 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、Automation Hub 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** の権限がある。

手順

1. ナビゲーションパネルから、**Execution Environments** → **Execution Environments** を選択します。
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. **More Actions** アイコン **!** 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。
6. (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。
7. **Save** をクリックします。

検証

- **Activity** タブをクリックし、最新の変更を確認します。

6.5. AUTOMATION CONTROLLER での認証情報の作成

パスワードまたはトークンで保護されたレジストリーからコンテナイメージをプルするには、Automation Controller で認証情報を作成する必要があります。

Ansible Automation Platform の以前のバージョンでは、実行環境イメージを格納するためにレジストリーをデプロイする必要がありました。Ansible Automation Platform 2.0 以降のシステムは、コンテナレジストリーがすでに稼働していると想定して動作します。実行環境イメージを格納するには、選択したコンテナレジストリーについてのみ認証情報を追加します。

手順

1. Automation Controller に移動します。
2. ナビゲーションパネルから **Resources** → **Credentials** を選択します。
3. **Add** をクリックして、新規の認証情報を作成します。
4. 承認用の **名前**、**説明**、および **組織** を入力します。
5. **認証情報のタイプ** を選択します。
6. **認証用 URL** を入力します。これはコンテナレジストリーのアドレスです。
7. コンテナレジストリーへのログインに必要な **ユーザー名** と **パスワードまたはトークン** を入力します。
8. オプション: SSL 検証を有効にするには、**Verify SSL** を選択します。
9. **Save** をクリックします。

第7章 コンテナリポジトリからのイメージのプル

Automation Hub 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。Automation Hub 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。

7.1. イメージのプル

Automation Hub 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。

前提条件

- プライベートコンテナリポジトリを表示およびプルする権限がある。

手順

1. パスワードまたはトークンで保護されたレジストリーからコンテナイメージをプルする必要がある場合は、イメージをプルする前に [Automation Controller](#) で [認証情報を作成](#) する必要があります。
2. ナビゲーションパネルから、**Execution Environments** → **Execution Environments** を選択します。
3. コンテナリポジトリを選択します。
4. **Pull this image** エントリーで、**Copy to clipboard** をクリックします。
5. 端末でコマンドを貼り付けます。

検証

- **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。

7.2. コンテナリポジトリからのイメージの同期

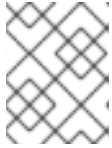
Automation Hub コンテナレジストリーからイメージをプルして、イメージをローカルマシンに同期できます。リモートコンテナレジストリーからイメージを同期するには、まずリモートレジストリーを設定する必要があります。

前提条件

プライベートコンテナリポジトリを表示およびプルする権限がある。

手順

1. ナビゲーションパネルから、**Execution Environments** → **Execution Environments** を選択します。
2. <https://registry.redhat.io> をレジストリーに追加します。
3. 認証に必要な認証情報を追加します。



注記

コンテナリレジストリーの中には、流量制御を積極的に行っているものもあります。**Advanced Options** で流量制御を設定します。

4. ナビゲーションパネルから、**Execution Environments** → **Execution Environments** を選択します。
5. ページヘッダーの **Add execution environment** をクリックします。
6. 取得元のレジストリーを選択します。**Name** フィールドには、ローカルレジストリーに表示されるイメージの名前が表示されます。



注記

Upstream name フィールドは、リモートサーバー上のイメージの名前です。たとえば、アップストリーム名が "alpine" に設定され、**Name** フィールドが "local/alpine" の場合、alpine イメージがリモートからダウンロードされ、"local/alpine" に名前が変更されます。

7. 追加または除外するタグのリストを設定します。イメージに多数のタグがあると、イメージの同期に時間がかかり、大量のディスク容量が使用されます。

関連情報

- レジストリーのリストについては、[Red Hat コンテナリレジストリーの認証](#) を参照してください。
- イメージをプルする際に使用するオプションは、[What is 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ソース) 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/system/** 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。