Red Hat Ansible Automation Platform 2.4

Red Hat Ansible Automation Platform のアップグレードおよび移行ガイド

Ansible Automation Platform のレガシーデプロイメントのアップグレードと移行

Last Updated: 2024-09-16

Red Hat Ansible Automation Platform 2.4 Red Hat Ansible Automation Platform のアップグレードおよび移行ガイド

Ansible Automation Platform のレガシーデプロイメントのアップグレードと移行

法律上の通知

概要

このガイドでは、Ansible Automation Platform の最新バージョンにアップグレードし、レガシー仮想環境をオートメーション実行環境に移行する方法を説明します。

目次

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 RED HAT ANSIBLE AUTOMATION PLATFORM 2.4 のアップグレード

インベントリーを設定し、インストールスクリプトを実行して Red Hat Ansible Automation Platform 2.4 にアップグレードします。その後、Ansible はデプロイメントを 2.4 にアップグレードします。Ansible Automation Platform 2.0 以前からアップグレードする場合は、2.4 との互換性を確保するために Ansible コンテンツを移行する必要があります。

1.1. ANSIBLE AUTOMATION PLATFORM のアップグレード

Ansible Automation Platform 2.1 以降からバージョン 2.4 にアップグレードするには、インストールパッケージをダウンロードしてから以下の手順を実行します。

- インストール環境に合わせてインベントリーを設定します。
- 現在の Ansible Automation Platform インストールで 2.4 インストールプログラムを実行します。

関連情報

- Red Hat Ansible Automation Platform 2.4 へのアップグレード

1.2. ANSIBLE AUTOMATION PLATFORM のレガシーアップグレード

Ansible Automation Platform 2.0 以前からバージョン 2.4 にアップグレードするには、互換性のために Ansible コンテンツを移行する必要があります。

以下の手順は、レガシーのアップグレードプロセスの概要を示しています。

- **awx-manage** コマンドを使用して、カスタムの仮想環境を自動化実行環境に複製します。
- ノードが最新の自動化メッシュ機能と互換性を持つように、サイドバイサイドアップグレードを実行して、分離されたレガシーノードから実行ノードにデータを移行します。
- 新しい自動化ハブ API トークンをインポートまたは生成します。
- **ansible-core** 2.15 との互換性のために、完全修飾コレクション名 (FQCN) を組み込むように Ansible コンテンツを再設定します。

関連情報

- 仮想環境を自動化実行環境に移行する
- 分離ノードの実行ノードへの移行
- Ansible コンテンツの移行

第2章 RED HAT ANSIBLE AUTOMATION PLATFORM 2.4 へのアップグレード

Red Hat Ansible Automation Platform をアップグレードするには、計画情報を確認してアップグレードが成功するようにします。次に、必要なバージョンの Ansible Automation Platform インストーラーをダウンロードし、環境を反映するようにインストールバンドル内のインベントリーファイルを設定してから、インストーラーを実行できます。

2.1. ANSIBLE AUTOMATION PLATFORM のアップグレード計画

アップグレードプロセスを開始する前に、以下の考慮事項を確認して、Ansible Automation Platform デプロイメントを計画し、準備してください。

Automation Controller

- 以前のバージョンから有効なライセンスがある場合でも、最新バージョンの自動化コントローラーにアップグレードする際に、認証情報またはサブスクリプションマニフェストを指定する必要があります。
- Red Hat Enterprise Linux および自動化コントローラーをアップグレードする必要がある場合は、まず自動化コントローラーデータをバックアップして復元する必要があります。
- クラスター形式のアップグレードでは、アップグレード前にインスタンスとインスタンスグループに特に留意が必要です。

関連情報

- サブスクリプションのインポート
- バックアップおよび復元
- クラスタリング

Automation Hub

- Ansible Automation Platform 2.4 にアップグレードする場合は、既存の自動化ハブ API トークンを追加するか、新規作成して既存トークンを無効にできます。
- Ansible Automation Platform をアップグレードすると、既存のコンテナーイメージは削除されます。これは、**setup.sh** スクリプトを使用して Ansible Automation Platform をアップグレードするときに、podman **system reset -f** が実行されるためです。これにより、Ansible Automation Platform ノード上のすべてのコンテナーイメージが削除され、インストーラーにバンドルされている新しい実行環境イメージがプッシュされます。Red Hat Ansible Automation Platform インストーラーセットアップスクリプトの実行 を参照してください。

関連情報

- インベントリーファイルの設定

Event-Driven Ansible Controller

- 現在 Event-Driven Ansible Controller を実行していて、Ansible Automation Platform 2.4 にアップグレードする際にそれをデプロイする予定の場合は、アップグレード前にすべての Event-Driven Ansible アクティベーションを無効にして、アップグレードプロセスの完了後に新しいア

クティベーションのみが実行されるようにすることを推奨します。これにより、孤立したコンテナーが以前のバージョンのアクティベーションを実行することが阻止されます。

2.2. RED HAT ANSIBLE AUTOMATION PLATFORM インストーラーの選択および取得

Red Hat Enterprise Linux 環境のインターネット接続に基づいて、必要な Ansible Automation Platform インストーラーを選択します。以下のシナリオを確認し、ニーズを満たす Red Hat Ansible Automation Platform インストーラーを決定してください。

注記

Red Hat カスタマーポータルで Red Hat Ansible Automation Platform インストーラーのダウンロードにアクセスするには、有効な Red Hat カスタマーアカウントが必要です。

インターネットアクセスを使用したインストール

Red Hat Enterprise Linux 環境がインターネットに接続している場合は、Ansible Automation Platform インストーラーを選択します。インターネットアクセスを使用してインストールすると、必要な最新のリポジトリー、パッケージ、および依存関係を取得します。Ansible Automation Platform インストーラーをセットアップするには、次のいずれかの方法を選択します。

tarball インストール

1. Red Hat Ansible Automation Platform のダウンロード ページに移動します。
2. **Ansible Automation Platform <latest-version> Setup** の **Download Now** をクリックします。
3. ファイルをデプロイメントします。

```
$ tar xvzf ansible-automation-platform-setup-<latest-version>.tar.gz
```

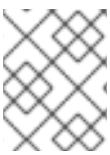
RPM インストール

1. Ansible Automation Platform インストーラーパッケージをインストールします。
   v.2.4 for RHEL 8 for x86_64

```
$ sudo dnf install --enablerepo=ansible-automation-platform-2.4-for-rhel-8-x86_64-rpms
ansible-automation-platform-installer
```

   v.2.4 for RHEL 9 for x86-64

```
$ sudo dnf install --enablerepo=ansible-automation-platform-2.4-for-rhel-9-x86_64-rpms
ansible-automation-platform-installer
```



注記

dnf install は、リポジトリーがデフォルトで無効になっているため、リポジトリーを有効にします。

RPM インストーラーを使用すると、ファイルは **/opt/ansible-automation-platform/installer** ディレクトリーに置かれます。

インターネットアクセスなしでのインストール

インターネットにアクセスできない場合や、オンラインリポジトリーから個別のコンポーネントおよび依存関係をインストールしたくない場合は、Red Hat Ansible Automation Platform の **Bundle** インストーラーを使用します。Red Hat Enterprise Linux リポジトリーへのアクセスは依然として必要です。その他の依存関係はすべて tar アーカイブに含まれます。

1. Red Hat Ansible Automation Platform のダウンロード ページに移動します。
2. **Ansible Automation Platform <latest-version> Setup Bundle**の **Download Now** をクリックします。
3. ファイルをデプロイメントします。

```
$ tar xvzf ansible-automation-platform-setup-bundle-<latest-version>.tar.gz
```

2.3. インベントリーファイルの設定

Red Hat Ansible Automation Platform インストールをアップグレードする前に、希望の設定に一致するように **inventory** ファイルを編集します。既存の Ansible Automation Platform デプロイメントと同じパラメーターを維持するか、使用環境に合わせてパラメーターを変更できます。

手順

1. インストールプログラムディレクトリーに移動します。

   バンドルのインストーラー

   ```
   $ cd ansible-automation-platform-setup-bundle-2.4-1-x86_64
   ```

   オンラインインストーラー

   ```
   $ cd ansible-automation-platform-setup-2.4-1
   ```

2. 編集する **inventory** ファイルを開きます。
3. **inventory** ファイルを変更して、新規ノードのプロビジョニング、ノードまたはグループのプロビジョニング解除、および自動化ハブ API トークンのインポートまたは生成を行います。環境に変更を加えない場合は、既存の Ansible Automation Platform 2.1 インストールから同じ **inventory** ファイルを使用できます。

注記

[automationhub] および **[automationcontroller]** ホストに到達可能な IP アドレスまたは完全修飾ドメイン名 (FQDN) を提供して、ユーザーが別のノードの Ansible Automation Hub からコンテンツを同期してインストールできるようにします。**localhost** は使用しないでください。**localhost** を使用している場合は、アップグレードがプリフライトチェックの一部として停止します。

クラスター内での新規ノードのプロビジョニング

- 以下のように **inventory** ファイルの既存ノードとともに新規ノードを追加します。

```
[controller]
clusternode1.example.com
clusternode2.example.com
clusternode3.example.com

[all:vars]
admin_password='password'

pg_host=''
pg_port=''

pg_database='<database_name>'
pg_username='<your_username>'
pg_password='<your_password>'
```

クラスター内のノードまたはグループのプロビジョニング解除

- **inventory** ファイルのノードまたはグループに **node_state-deprovision** を追加します。

API トークンのインポートと生成

Red Hat Ansible Automation Platform 2.0 以前から Red Hat Ansible Automation Platform 2.1 以降にアップグレードする場合、既存の Automation Hub API トークンを使用することも、新しいトークンを生成することもできます。インベントリーファイルで、Red Hat Ansible Automation Platform インストーラーのセットアップスクリプト **setup.sh** を実行する前に、以下のいずれかのフィールドを編集します。

- 以下のように **automationhub_api_token** フラグを使用して既存の API トークンをインポートします。

```
automationhub_api_token=<api_token>
```

- 以下のように **generate_automationhub_token** フラグを使用して、新しい API トークンを生成し、既存のトークンを無効にします。

```
generate_automationhub_token=True
```

関連情報

- Red Hat Ansible Automation Platform インストールガイド
- 個別ノードまたはインスタンスグループのプロビジョニング解除

2.4. RED HAT ANSIBLE AUTOMATION PLATFORM インストーラー設定スクリプトの実行

inventory ファイルの更新が完了したら、設定スクリプトを実行できます。

手順

1. **setup.sh** スクリプトを実行します。

```
$ ./setup.sh
```

```
-
```

インストールが開始します。

第3章 自動化実行環境への移行

3.1. 自動化実行環境にアップグレードする理由

Red Hat Ansible Automation Platform 2.4 には、自動化実行環境が導入されています。自動化実行環境は、単一のコンテナー内で Ansible Automation を実行するために必要なすべてのものを含めることにより、Ansible の管理を容易にするコンテナーイメージです。自動化実行環境には、以下が含まれます。

- RHEL UBI 8
- ansible-core 2.14 以降
- Python 3.9 以降
- Ansible コンテンツコレクション
- Python またはバイナリーの依存関係のコレクション

これらの要素を含めることで、Ansible はプラットフォーム管理者は、自動化が実行される環境を標準化して定義、構築、および配布できるようになります。

新しい自動化実行環境により、管理者がカスタムプラグインや自動化コンテンツを作成する必要がなくなりました。管理者は、これまでよりも短時間で、サイズのより小さい自動化実行環境を起動できるようになります。

すべてのカスタム依存関係は、管理およびデプロイメントフェーズではなく、開発フェーズで定義されるようになりました。コントロールプレーンから分離することで、開発サイクルの高速化、スケーラビリティ、信頼性、および環境間での移植性が実現します。自動化実行環境により、Ansible Automation Platform を分散アーキテクチャーに移行して、管理者が組織全体で自動化を拡張できるようになります。

3.2. レガシー VENVS の自動化実行環境への移行について

古いバージョンの自動化コントローラーからバージョン 4.0 以降にアップグレードすると、コントローラーは、組織、インベントリー、およびジョブテンプレートに関連付けられた以前のバージョンの仮想環境を検出して、新しい自動化実行環境モデルに移行する必要があることを通知します。自動化コントローラーを新たにインストールすると、インストール中に 2 つの virtualenv が作成されます。1 つはコントローラーを実行し、もう 1 つは Ansible を実行します。従来の仮想環境と同様に、自動化実行環境では、コントローラーを安定した環境で実行できますが、Playbook を実行するために必要に応じて、自動化実行環境にモジュールを追加または更新できます。

自動化実行環境でのセットアップを、以前のカスタム仮想環境から新しい自動化実行環境に移行することで複製できます。このセクションの **awx-manage** コマンドを使用して、以下を実行します。

- 現在のすべてのカスタムの仮想環境とそのパスをリスト表示する (**list_custom_venvs**)
- 特定のカスタムの仮想環境に依存するリソースの表示する (**custom_venv_associations**)
- 特定のカスタム仮想環境を、自動化実行環境への移行に使用できる形式にエクスポートする (**export_custom_venv**)。

以下のワークフローでは、**awx-manage** コマンドを使用して、古い venvs から自動化実行環境に移行する方法を説明します。

3.3. 仮想環境を自動化実行環境に移行

Red Hat Ansible Automation Platform 2.0 および Automation Controller 4.0 にアップグレードしたら、移行プロセスの追加手順について、以下のセクションを使用します。

3.3.1. カスタムの仮想環境の構築

awx-manage コマンドを使用して、Automation Controller インスタンスの仮想環境をリスト表示できます。

手順

1. 自動コントローラーインスタンスに SSH 接続して実行します。

```
$ awx-manage list_custom_venvs
```

検出された仮想環境のリストが表示されます。

```
# Discovered virtual environments:
/var/lib/awx/venv/testing
/var/lib/venv/new_env

To export the contents of a virtual environment, re-run while supplying the path as an argument:
awx-manage export_custom_venv /path/to/venv
```

3.3.2. カスタムの仮想環境に関連付けられたオブジェクトの表示

awx-manage コマンドを使用して、カスタムの仮想環境に関連付けられた組織、ジョブ、およびインベントリーソースを表示します。

手順

1. 自動コントローラーインスタンスに SSH 接続して実行します。

```
$ awx-manage custom_venv_associations /path/to/venv
```

関連付けられたオブジェクトのリストが表示されます。

```
inventory_sources:
- id: 15
  name: celery
job_templates:
- id: 9
  name: Demo Job Template @ 2:40:47 PM
- id: 13
  name: elephant
organizations
- id: 3
  name: alternating_bongo_meow
- id: 1
  name: Default
projects: []
```

3.3.3. エクスポートするカスタムの仮想環境の選択

awx-manage export_custom_venv コマンドを使用して、エクスポートするカスタムの仮想環境を選択します。

手順

1. 自動コントローラーインスタンスに SSH 接続して実行します。

```
$ awx-manage export_custom_venv /path/to/venv
```

このコマンドにより、指定した仮想環境に含まれる **pip freeze** が表示されます。この情報は、Ansible Builder の **requirements.txt** ファイルにコピーして、新しい自動化実行環境イメージを作成するために使用できます。

```
numpy==1.20.2
pandas==1.2.4
python-dateutil==2.8.1
pytz==2021.1
six==1.16.0

To list all available custom virtual environments run:
awx-manage list_custom_venvs
```

注記

awx-manage list_custom_venvs を実行して出力を減らしたときに **-q** フラグを渡します。

第4章 分離ノードの実行ノードへの移行

バージョン 1.x から最新バージョンの Red Hat Automation Platform にアップグレードするには、プラットフォーム管理者が分離されたレガシーノードから実行ノードに、データを移行する必要があります。この移行は、自動化メッシュのデプロイに必要です。

このガイドでは、サイドバイサイド移行を実行する方法を説明します。これにより、移行プロセス中に元の自動化環境のデータが変更されないようになります。

移行プロセスには、次の手順が含まれます。

1. アップグレード設定を確認します。
2. 元のインスタンスをバックアップします。
3. サイドバイサイドアップグレード用に新しいインスタンスをデプロイします。
4. Ansible コントローラーを使用して、新しいインスタンスにインスタンスグループを再作成します。
5. 元のバックアップを新しいインスタンスに復元します。
6. 実行ノードを設定し、インスタンスを Red Hat Ansible Automation Platform 2.4 にアップグレードします。
7. アップグレードされたコントローラーインスタンスを設定します。

4.1. ANSIBLE AUTOMATION PLATFORM をアップグレードするための前提条件

Ansible Automation Platform のアップグレードを開始する前に、環境が次のノードと設定の要件を満たしていることを確認してください。

4.1.1. ノードの要件

Ansible Automation Platform のアップグレードプロセスに関係するノードには、次の仕様が必要です。

- コントローラーノード、データベースノード、実行ノード、およびホップノード用の 16GB の RAM
- コントローラーノード、データベースノード、実行ノード、およびホップノード用の 4 つの CPU
- データベースノード用に 150 GB 以上のディスク容量
- データベースノード以外のノード用に 40 GB 以上のディスク容量
- DHCP 予約では、無限リースを使用して、静的 IP アドレスを持つクラスターをデプロイメント
- すべてのノードの DNS レコード
- すべてのノードで Red Hat Enterprise Linux 8 以降 64 ビット (x86) がインストール済み
- Chrony はすべてのノードに設定済み
- すべてのコンテンツ依存関係については Python3.9 以降

4.1.2. 自動化コントローラーの設定要件

Ansible Automation Platform のアップグレードプロセスを進める前に、次の自動化コントローラーの設定が必要です。

Chrony を使用した NTP サーバーの設定

クラスター内の各 Ansible Automation Platform ノードは NTP サーバーにアクセスできる必要があります。**chronyd** を使用して、システムクロックを NTP サーバーと同期します。これにより、ノード間の日付と時刻が同期していない場合でも、検証が必要な SSL 証明書を使用するクラスターノードが失敗しなくなります。

これは、アップグレードされた Ansible Automation Platform クラスターで使用されるすべてのノードで必要です。

1. **chrony** をインストールします。

```
# dnf install chrony --assumeyes
```

2. テキストエディターを使用して **/etc/chrony.conf** を開きます。

3. パブリックサーバープールセクションを見つけて、適切な NTP サーバーアドレスが含まれるように変更します。必要なサーバーは 1 台だけですが、推奨は 3 台です。'iburst' オプションを追加して、サーバーと適切に同期するのにかかる時間を短縮します。

```
# Use public servers from the pool.ntp.org project.
# Please consider joining the pool (http://www.pool.ntp.org/join.html).
server <ntp-server-address> iburst
```

4. **/etc/chrony.conf** ファイル内に変更を保存します。

5. ホストを起動し、**chronyd** デーモンを有効にします。

```
# systemctl --now enable chronyd.service
```

6. **chronyd** デーモンのステータスを確認します。

```
# systemctl status chronyd.service
```

すべてのノードでの Red Hat サブスクリプションのアタッチ

Red Hat Ansible Automation Platform では、すべてのノードに有効なサブスクリプションをアタッチする必要があります。次のコマンドを実行して、現在のノードに Red Hat サブスクリプションがあることが確認できます。

```
# subscription-manager list --consumed
```

ノードにアタッチされている Red Hat サブスクリプションがない場合の詳細は、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。

sudo 権限を持つ root 以外のユーザーの作成

Ansible Automation Platform をアップグレードする前に、デプロイプロセスの sudo 権限を持つ root 以外のユーザーを作成することを推奨します。このユーザーは以下で使用されます。

- SSH 接続
- インストール時中のパスワードレス認証
- 特権昇格 (sudo) 権限

次の例では、**ansible** を使用してこのユーザーに名前を付けています。アップグレードされた Ansible Automation Platform クラスターで使用されるすべてのノードで、**ansible** という名前で root 以外のユーザーを作成し、SSH キーを生成します。

1. 非 root ユーザーを作成します。

```
# useradd ansible
```

2. ユーザーのパスワードを設定します。

```
# passwd ansible 1
Changing password for ansible.
Old Password:
New Password:
Retype New Password:
```

1 別の名前を使用している場合は、**ansible** を手順 1 の root 以外のユーザーに置き換えます。

3. ユーザーとして **ssh** キーを生成します。

```
$ ssh-keygen -t rsa
```

4. **sudo** を使用する場合にパスワードを要求されないようにします。

```
# echo "ansible ALL=(ALL) NOPASSWD:ALL" | sudo tee -a /etc/sudoers.d/ansible
```

SSH キーをすべてのノードにコピー

ansible ユーザーを作成したら、アップグレードされた Ansible Automation Platform クラスターで使用されているすべてのノードに **ssh** キーをコピーします。これにより、Ansible Automation Platform のインストールが実行されたときに、パスワードなしですべてのノードへの **ssh** が可能になります。

```
$ ssh-copy-id ansible@node-1.example.com
```

注記

クラウドプロバイダー内で実行している場合は、代わりに、すべてのノードに **ansible** ユーザーの公開鍵を含む **~/.ssh/authorized_keys** ファイルを作成し、**authorized_keys** ファイルのパーミッションは、所有者 (**ansible**) だけに読み取りおよび書き込み権限 (パーミッション 600) を設定します。

ファイアウォールの設定

アップグレードされた Ansible Automation Platform クラスターで使用されるすべてのノードでファイアウォール設定を指定して、Ansible Automation Platform を正常にアップグレードするために適切な

サービスとポートへのアクセスを許可します。Red Hat Enterprise Linux 8 以降の場合は、**firewalld** デーモンを有効にして、すべてのノードに必要なアクセスを有効にします。

1. **firewalld** パッケージをインストールします。

```
# dnf install firewalld --assumeyes
```

2. **firewalld** サービスを開始します。

```
# systemctl start firewalld
```

3. **firewalld** サービスを有効にします。

```
# systemctl enable --now firewalld
```

4.1.3. Ansible Automation Platform の設定要件

Ansible Automation Platform のアップグレードプロセスを進める前に、次の Ansible Automation Platform 設定が必要です。

実行ノードとホップノードのファイアウォール設定

Red Hat Ansible Automation Platform インスタンスをアップグレードした後に、自動メッシュ機能を有効にするために、メッシュノード (実行ノードおよびホップノード) に自動メッシュポートを追加します。すべてのノードのメッシュネットワークに使用されるデフォルトのポートは **27199/tcp** です。インベントリーファイル内の各ノードの変数として **recptor_listener_port** を指定することにより、異なるポートを使用するようにメッシュネットワークを設定できます。

ホップおよび実行ノード内で、インストールに使用する **firewalld** ポートを設定します。

1. **firewalld** が実行されていることを確認します。

```
$ sudo systemctl status firewalld
```

2. コントローラーデータベースノードに **firewalld** ポートを追加します (例: ポート 27199)。

```
$ sudo firewall-cmd --permanent --zone=public --add-port=27199/tcp
```

3. **firewalld** をリロードします。

```
$ sudo firewall-cmd --reload
```

4. ポートが開いていることを確認します。

```
$ sudo firewall-cmd --list-ports
```

4.2. ANSIBLE AUTOMATION PLATFORM インスタンスのバックアップ

backup_dir フラグを指定して **.setup.sh** スクリプトを実行し、既存の Ansible Automation Platform インスタンスをバックアップします。これにより、現在の環境のコンテンツと設定が保存されます。

1. **ansible-tower-setup-latest** ディレクトリーに移動します。

2. 以下の例に従って、**./setup.sh** スクリプトを実行します。

```
$ ./setup.sh -e 'backup_dir=/ansible/mybackup' -e 'use_compression=True' @credentials.yml
-b (1)(2)
```

(1) **backup_dir** は、バックアップを保存するディレクトリーを指定します。

(2) **@ credentials.yml** は、パスワード変数およびその値を **ansible-vault** で暗号化して渡します。

バックアップが成功すると、バックアップファイルが **/ansible/mybackup/tower-backup-latest.tar.gz** に作成されます。

このバックアップは、後でコンテンツを古いインスタンスから新しいインスタンスに移行するために必要になります。

4.3. サイドバイサイドアップグレード用の新規インスタンスのデプロイ

サイドバイサイドアップグレードプロセスを続行するには、同じインスタンスグループ設定で Ansible Tower 3.8.x の 2 番目のインスタンスをデプロイします。この新しいインスタンスは、元のインスタンスからコンテンツおよび設定を受け取り、後で Red Hat Ansible Automation Platform 2.4 にアップグレードされます。

4.3.1. Ansible Tower の新規インスタンスのデプロイ

新しい Ansible Tower インスタンスをデプロイするには、次の手順を実行します。

1. Ansible Tower インストーラーページ に移動して、元の Tower インスタンスに対応する Tower インストーラーバージョンをダウンロードします。

2. インストーラーに移動し、テキストエディターを使用して **inventory** ファイルを開き、Tower インストール用の **inventory** ファイルを設定します。

   a. Tower を設定したら、**isolated_group** または **instance_group** を含むフィールドをすべて削除します。

注記

Ansible Automation Platform インストーラーを使用した Tower のインストールにおける特定のインストールシナリオの詳細は、Ansible Automation Platform インストールガイド を参照してください。

3. **setup.sh** スクリプトを実行して、インストールを開始します。

新しいインスタンスがインストールされたら、元の Tower インスタンスのインスタンスグループと一致するように Tower 設定を指定します。

4.3.2. 新しいインスタンスでのインスタンスグループの再作成

新しいインスタンスでインスタンスグループを再作成するには次の手順を実行します。

注記

元の Tower インスタンスのすべてのインスタンスグループをメモします。新しいインスタンスでこれらのグループを再作成する必要があります。

1. Tower の新しいインスタンスにログインします。
2. ナビゲーションペインで、**Administration → Instance groups** を選択します。
3. **Create instance group** をクリックします。
4. 元のインスタンスのインスタンスグループと同じ **Name** を入力し、**Save** をクリックします
5. 元のインスタンスのインスタンスグループがすべて再作成されるまで繰り返します。

4.4. 新しいインスタンスへのバックアップの復元

restore_backup_file フラグを指定して **./setup.sh** スクリプトを実行すると、コンテンツが元の 1.x インスタンスのバックアップファイルから新しいインスタンスに移行されます。これにより、すべてのジョブ履歴、テンプレート、およびその他の Ansible Automation Platform 関連のコンテンツが効果的に移行されます。

手順

1. 以下のコマンドを実行します。

```
$ ./setup.sh -r -e 'restore_backup_file=/ansible/mybackup/tower-backup-latest.tar.gz' -e
'use_compression=True' -e @credentials.yml -r -- --ask-vault-pass (1) (2) (3)
```

(1) **restore_backup_file** は、Ansible Automation Platform バックアップデータベースの場所を指定します

(2) バックアッププロセス中に圧縮が使用されているため、**use_compression** は **True** に設定されています

(3) **-r** は、データベースの復元オプションを **True** に設定します

2. 新しい RHEL 8 Tower 3.8 インスタンスにログインして、元のインスタンスのコンテンツが復元されているかどうかを確認します。
   a. **Administration → Instance Groups** に移動します。再作成されたインスタンスグループには、元のインスタンスの **Total Jobs** が含まれているはずです。
   b. サイドナビゲーションパネルを使用して、ジョブ、テンプレート、インベントリー、クレデンシャル、ユーザーなどのコンテンツが元のインスタンスからインポートされていることを確認します。

これで、元のインスタンスの全 Ansible コンテンツを含む Ansible Tower の新しいインスタンスができました。

この新しいインスタンスを Ansible Automation Platform 2.4 にアップグレードして、元のインスタンスを上書きせずに以前のすべてのデータを保持できるようにします。

4.5. ANSIBLE AUTOMATION PLATFORM 2.4 へのアップグレード

Ansible Tower のインスタンスを Ansible Automation Platform 2.4 にアップグレードするには、**inventory** ファイルを元の Tower インスタンスから新しい Tower インスタンスにコピーし、インストーラーを実行します。Red Hat Ansible Automation Platform インストーラーは、2.4 より前のバージョンを検出し、アップグレードされたインベントリーファイルを提供して、アップグレードプロセスを続行します。

1. Red Hat Ansible Automation Platform のダウンロード ページから Red Hat Ansible Automation Platform の最新のインストーラーをダウンロードします。
2. ファイルをデプロイメントします。

```
$ tar xvzf ansible-automation-platform-setup-<latest_version>.tar.gz
```

3. Ansible Automation Platform のインストールディレクトリーに移動します。

```
$ cd ansible-automation-platform-setup-<latest_version>/
```

4. 元のインスタンスから最新のインストーラーのディレクトリーに **inventory** ファイルをコピーします。

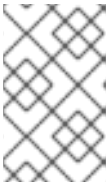
```
$ cp ansible-tower-setup-3.8.x.x/inventory ansible-automation-platform-
setup-<latest_version>
```

5. **setup.sh** スクリプトを実行します。

```
$ ./setup.sh
```

セットアップスクリプトは一時停止し、"pre-2.x" インベントリーファイルが検出されたことを示しますが、**inventory.new.ini** という新しいファイルが提供され、元のインスタンスのアップグレードを続行できます。

6. テキストエディターで **inventory.new.ini** を開きます。



注記

インストーラーは、セットアップスクリプトを実行することで、tower の名前を automationcontroller に変更するなど、元のインベントリーファイルからいくつかのフィールドを変更しました。

7. 新しく生成された **inventory.new.ini** ファイルを更新して、関連する変数、ノード、および関連するノード間ピア接続を割り当てて、自動化メッシュを設定します。

注記

自動化メッシュトポロジーの設計は、環境の自動化のニーズによって異なります。本書では、考えられるシナリオの設計をすべて提供することはしていません。以下は、自動化メッシュ設計の例です。

ホップノードを利用する 3 つのノードで構成される標準のコントロールプレーンを含むインベントリーファイルの例:

```
[automationcontroller]
control-plane-1.example.com
```

```
control-plane-2.example.com
control-plane-3.example.com

[automationcontroller:vars]
node_type=control (1)
peers=execution_nodes (2)


[execution_nodes]
execution-node-1.example.com peers=execution-node-2.example.com
execution-node-2.example.com peers=execution-node-3.example.com
execution-node-3.example.com peers=execution-node-4.example.com
execution-node-4.example.com peers=execution-node-5.example.com node_type=hop
execution-node-5.example.com peers=execution-node-6.example.com node_type=hop (3)
execution-node-6.example.com peers=execution-node-7.example.com
execution-node-7.example.com

[execution_nodes:vars]
node_type=execution
```

(1) プロジェクトとインベントリーの更新やシステムジョブを実行するが、通常のジョブを実行しない、制御ノードを指定します。これらのノードでは実行機能は無効になります。

(2) **execution_nodes** グループ内のノード間接続のピア関係を指定します。

(3) トラフィックを他の実行ノードにルーティングするホップノードを指定します。ホップノードは自動化を実行できません。

8. 自動化ハブ API トークンをインポートまたは生成します。
   - **automationhub_api_token** フラグを使用して既存の API トークンをインポートします。

     ```
     automationhub_api_token=<api_token>
     ```

   - **generate_automationhub_token** フラグを **True** に設定して、新しい API トークンを生成し、既存のトークンを無効にします。

     ```
     generate_automationhub_token=True
     ```

9. 自動化メッシュ用に **inventory.new.ini** の設定が完了したら、**inventory.new.ini** を使用してセットアップスクリプトを実行します。

   ```
   $ ./setup.sh -i inventory.new.ini -e @credentials.yml -- --ask-vault-pass
   ```

10. インストールが完了したら、すべての自動化コントローラーノードで Ansible Automation Platform ダッシュボード UI にログインして、Ansible Automation Platform が正常にインストールされたことを確認します。

関連情報

- Ansible Automation Platform インストーラーの使用に関する一般的な情報は、Red Hat Ansible Automation Platform インストールガイド を参照してください。

4.6. アップグレードされた ANSIBLE AUTOMATION PLATFORM の設定

4.6.1. 自動化コントローラーインスタンスグループの設定

Red Hat Ansible Automation Platform インスタンスをアップグレードした後に、自動化コントローラー UI で設定を指定して、元のインスタンスを対応するインスタンスグループに関連付けます。

1. 新しい Controller インスタンスにログインします。
2. クレデンシャル、ジョブ、インベントリーなどの古いインスタンスのコンテンツが、コントローラーインスタンスに表示されるようになります。
3. **Administration → Instance Groups** に移動します。
4. インスタンスグループをクリックして実行ノードを関連付け、**Instances** タブをクリックします。
5. **Associate** をクリックします。このインスタンスグループに関連付けるノードを選択し、**保存** をクリックします。
6. デフォルトのインスタンスを変更して、新しい実行ノードの関連付けを解除することもできます。

第5章 ANSIBLE コンテンツの移行

ansible-core バージョンから **ansible-core** 2.13 に移行する場合は、各バージョン間の変更と更新について理解できるように Ansible Core 移植ガイド を確認することを検討してください。Ansible Core 移植ガイドを確認する場合は、ガイドの左上の列にある最新バージョンの **ansible-core** または **devel** を選択してください。

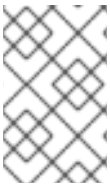
完全にサポートおよび認定された Ansible コンテンツコレクションのリストについては、console.redhat.com の Ansible Automation Hub を参照してください。

5.1. ANSIBLE コレクションのインストール

以前のバージョンの Ansible から最新バージョンへの移行の一環として、使用しているモジュールを含むコレクションを見つけてダウンロードする必要があります。コレクションのリストを見つけたら、次のいずれかのオプションを使用してコレクションをローカルに含めることができます。

1. **ansible-builder** を使用して、コレクションをランタイム環境または実行環境にダウンロードしてインストールします。

2. Automation Controller プロジェクトのインストールロールとコレクションの 'requirements.yml' ファイルを更新します。こうすることで、Automation Controller でプロジェクトを同期するたびに、ロールとコレクションがダウンロードされます。



注記

多くの場合、アップストリームコレクションとダウンストリームコレクションは同じにすることができますが、必ず Automation Hub から認定コレクションをダウンロードしてください。

5.2. ANSIBLE PLAYBOOK とロールの CORE 2.13 への移行

非コレクションベースのコンテンツからコレクションベースのコンテンツに移行する場合は、予期しない動作を回避するために、Playbook とロールで完全修飾コレクション名 (FQCN) を使用する必要があります。

FQCN を使用した Playbook の例:

```
- name: get some info
  amazon.aws.ec2_vpc_net_info:
    region: "{{ec2_region}}"
  register: all_the_info
  delegate_to: localhost
  run_once: true
```

ansible-core モジュールを使用していて、別のコレクションからモジュールを呼び出していない場合は、**FQCNansible.builtin.copy** を使用する必要があります。

FQCN を使用したモジュールの例:

```
- name: copy file with owner and permissions
  ansible.builtin.copy:
  src: /srv/myfiles/foo.conf
  dest: /etc/foo.conf
```

```
  owner: foo
  group: foo
  mode: '0644'
```

5.3. PLAYBOOK 例の変換

例

これは、ジョブの実行中にファイルの読み取りと書き込みをできるようにする **/mydata** という共有ディレクトリーの例です。これは、オートメーションの実行に使用する実行ノードにすでに存在している必要がある点に注意してください。

このジョブを実行するには **aape1.local** 実行ノードをターゲットにします。基盤となるホストにはすでにこれが設定されているからです。

```
[awx@aape1 ~]$ ls -la /mydata/
total 4
drwxr-xr-x.  2 awx  awx   41 Apr 28 09:27 .
dr-xr-xr-x. 19 root root 258 Apr 11 15:16 ..
-rw-r--r--.  1 awx  awx   33 Apr 11 12:34 file_read
-rw-r--r--.  1 awx  awx    0 Apr 28 09:27 file_write
```

簡単な Playbook を使って、アクセスを許可するために定義されたスリープでオートメーションを起動し、プロセスを理解して、ファイルの読み書きを示します。

```
# vim:ft=ansible:
```

```
- hosts: all
  gather_facts: false
  ignore_errors: yes
  vars:
    period: 120
    myfile: /mydata/file
  tasks:
    - name: Collect only selected facts
      ansible.builtin.setup:
        filter:
          - 'ansible_distribution'
          - 'ansible_machine_id'
          - 'ansible_memtotal_mb'
          - 'ansible_memfree_mb'
    - name: "I'm feeling real sleepy..."
      ansible.builtin.wait_for:
        timeout: "{{ period }}"
      delegate_to: localhost
    - ansible.builtin.debug:
        msg: "Isolated paths mounted into execution node: {{ AWX_ISOLATIONS_PATHS }}"
    - name: "Read pre-existing file..."
      ansible.builtin.debug:
        msg: "{{ lookup('file', '{{ myfile }}_read'
    - name: "Write to a new file..."
      ansible.builtin.copy:
        dest: "{{ myfile }}_write"
        content: |
```

```
        This is the file I've just written to.

    - name: "Read written out file..."
      ansible.builtin.debug:
        msg: "{{ lookup('file', '{{ myfile }}_write') }}"
```

Ansible Automation Platform 2 ナビゲーションパネルから **Settings** を選択します。次に、**Jobs** オプションから **Job settings** を選択します。

分離されたジョブを公開するパス:

```
[
"/mydata:/mydata:rw"
]
```

ボリュームマウントはコンテナー内で同じ名前でマップされ、読み取り/書き込み機能を備えています。これは、ジョブテンプレートを起動するときに使用されます。

実行ごとにスリープ期間を調整できるように、**起動時のプロンプト** を **extra_vars** に設定する必要があります。デフォルトは 30 秒です。

起動され、**wait_for** モジュールがスリープのために呼び出されたら、実行ノードに行き、何が実行されているかを確認できます。

実行が正常に完了したことを確認するには、次のコマンドを実行してジョブの出力を取得します。

```
$ podman exec -it 'podman ps -q' /bin/bash
bash-4.4#
```

これで、実行中の実行環境コンテナーに入りました。

権限を見ると、**awx** が root になっていることがわかりますが、ユーザーをサンドボックスに似たカーネル名前空間にマップするルートレス Podman を使用しているため、これはスーパーユーザーのような実際の root ではありません。shadow-utils 用のルートレス Podman の仕組みに関する詳細は、How does rootless Podman work? を参照してください。

```
bash-4.4# ls -la /mydata/
Total 4
drwxr-xr-x. 2 root root 41 Apr 28 09:27 .
dr-xr-xr-x. 1 root root 77 Apr 28 09:40 ..
-rw-r—--r–. 1 root root 33 Apr 11 12:34 file_read
-rw-r—--r–. 1 root root  0 Apr 28 09:27 file_write
```

結果を確認すると、このジョブは失敗しました。その理由を理解するには、残りの出力を調べる必要があります。

```
TASK [Read pre-existing file…]****************************** 10:50:12
ok: [localhost] =>  {
    “Msg”: “This is the file I am reading in.”

TASK {Write to a new file...}******************************** 10:50:12
An exception occurred during task execution. To see the full traceback, use -vvv. The error was:
PermissionError: [Errno 13] Permission denied: b'/mydata/.ansible_tmpazyqyqdrfile_write' -> b'
/mydata/file_write'
Fatal: [localhost]: FAILED! => {"changed": false, :checksum":
```

```
"9f576o85d584287a3516ee8b3385cc6f69bf9ce", "msg": "Unable to make
b'/root/.ansible/tmp/anisible-tim-1651139412.9808054-40-91081834383738/source' into
/mydata/file_write, failed final rename from b'/mydata/.ansible_tmpazyqyqdrfile_write': [Errno 13]
Permission denied: b'/mydata/.ansible_tmpazyqyqdrfile_write' -> b'/mydata/file_write}
...ignoring

TASK [Read written out file...] ****************************** 10:50:13
Fatal: [localhost]: FAILED: => {"msg": "An unhandled exception occurred while running the lookup
plugin 'file'. Error was a <class 'ansible.errors.AnsibleError;>, original message: could not locate file in
lookup: /mydate/file_write. Vould not locate file in lookup: /mydate/file_write"}
...ignoring
```

:rw が設定されているにもかかわらずジョブは失敗したため、書き込み機能が必要です。プロセスは既存のファイルを読み取ることはできましたが、書き出すことはできませんでした。これは、コンテナーにマウントされたボリュームコンテンツに適切なラベルを付ける必要がある SELinux 保護によるものです。ラベルがない場合、SELinux はコンテナー内でのプロセスの実行を阻止する可能性があります。OS によって設定されたラベルは、Podman によって変更されることはありません。詳細は、Podman ドキュメントを参照してください。

これはよくある誤解である可能性があります。デフォルトを **:z** に設定しました。これにより、Podman は共有ボリューム上のファイルオブジェクトのラベルを変更するように指示されます。

したがって、**:z** を追加することも、追加しないこともできます。

分離されたジョブを公開するパス:

```
[
   "/mydata:/mydata"
]
```

Playbook は期待どおりに機能するようになりました。

```
PLAY [all] ******************************************************** 11:05:52
TASK [I'm feeling real sleepy. . .] ************************** 11:05:52
ok: [localhost]
TASK [Read pre-existing file...] ***************************** 11:05:57
ok: [localhost] => {
 "Msg": "This is the file I'm reading in."
}
TASK [Write to a new file…] ********************************** 11:05:57
ok: [localhost]
TASK [Read written out file…] ******************************** 11:05:58
ok: [localhost] => {
      “Msg”: “This is the file I’ve just written to.”
```

基盤となる実行ノードのホストに戻ると、新しく書き出された内容があります。

注記

Red Hat OpenShift 内で自動化ジョブを起動するためにコンテナーグループを使用している場合、同じパスをその環境に公開するように Ansible Automation Platform 2 に指示することもできますが、設定でデフォルトを **On** に切り替える必要があります。

有効にすると、これは、実行に使用される Pod 仕様内に **volumeMounts** および **volumes** として注入します。以下に例を示します。

```
apiVersion: v1
kind: Pod
Spec:
   containers:
   - image: registry.redhat.io/ansible-automation-platform-24/ee-minimal-rhel8
  args:
    - ansible runner
    - worker
    - –private-data-dir=/runner
  volumeMounts:
mountPath: /mnt2
name: volume-0
readOnly: true
mountPath: /mnt3
name: volume-1
readOnly: true
mountPath: /mnt4
name: volume-2
readOnly: true
volumes:
hostPath:
  path: /mnt2
  type: “”
name: volume-0
hostPath:
  path: /mnt3
  type: “”
name: volume-1
hostPath:
  path: /mnt4
  type: “”
name: volume-2
```

実行中のコンテナー内のストレージはオーバーレイファイルシステムを使用しています。tmpfs がアンマウントされるのと同様に、実行中のコンテナー内の変更はジョブの完了後に破棄されます。

第6章 AAP2 用の PLAYBOOK の変換

Ansible Automation Platform 2 とそのコンテナー化された実行環境では、**localhost** の使用方法が変更されました。Ansible Automation Platform の以前のバージョンでは、**localhost** に対してジョブが実行され、これは基盤となる Automation Controller ホスト上で実行されていました。これは、データと永続的なアーティファクトを保存するために使用できます。

Ansible Automation Platform 2 では、localhost はコンテナー内で実行されていることを意味しますが、これは本質的に一時的なものです。**Localhost** は、特定のホストに関連付けられなくなり、ポータブルな実行環境では、適切な環境とソフトウェアの前提条件がすでに実行環境コンテナーに組み込まれていれば、どこでも実行できることになります。

6.1. 自動実行からのデータの永続化

ローカル Automation Controller ファイルシステムは、データをそのホストに結び付けるため、逆効果であると考えてください。マルチノードクラスターがある場合、毎回異なるホストに接続することができ、これにより、相互に依存するワークフローを作成中で、ディレクトリーが作成済みの場合は、問題が発生する可能性があります。たとえば、ディレクトリーが 1 つのノードでのみ作成され、別のノードで Playbook が実行されている場合、矛盾する結果となります。

解決策は、Amazon S3、Gist、またはデータをデータエンドポイントに rsync するロールなど、何らかの形式の共有ストレージソリューションを使用することです。

ランタイムにデータまたは設定をコンテナーに注入するオプションが存在します。これは、Automation Controller の分離されたジョブパスオプションを使用して実現できます。

これにより、ランタイムにディレクトリーとファイルを実行環境にマウントする方法が提供されます。これは、ansible-runner を使用して自動化メッシュを Podman コンテナーに注入し、自動化を開始することにより、自動化メッシュを通じて実現されます。以下は、分離されたジョブパスを使用するユースケースの一部を示しています。

- SSL 証明書を実行環境に組み込むのではなく、ランタイムに提供します。
- SSH 設定などのランタイム設定データを渡しますが、自動化中に使用したいものであれば何でもかまいません。
- オートメーションの実行前、実行中、実行後に使用されるファイルの読み取りと書き込み。

使用するには次のような注意点があります。

- ボリュームマウントは、自動化実行が可能なすべてのノード (つまり、ハイブリッドコントロールプレーンノードとすべての実行ノード) に事前に存在する必要があります。
- SELinux が有効になっている場合 (Ansible Automation Platform のデフォルト)、ファイル権限に注意してください。
    - ルートレス Podman は OCP ベース以外のインストールで実行されるため、これは重要です。

注意事項を慎重に確認する必要があります。ルートレス Podman と Podman ボリュームマウントのランタイムオプション、分離されたジョブパスの [:OPTIONS] 部分についてよく読むことを強く推奨します。これが、Ansible Automation Platform 2 内で使用されるものだからです。

関連情報

- Understanding rootless Podman

- Podman volume mount runtime options

6.1.1. Playbook 例の変換

例

これは、ジョブの実行中にファイルの読み取りと書き込みをできるようにする **/mydata** という共有ディレクトリーの例です。これは、オートメーションの実行に使用する実行ノードにすでに存在している必要がある点に注意してください。

このジョブを実行するには **aape1.local** 実行ノードをターゲットにします。基盤となるホストにはすでにこれが設定されているからです。

```
[awx@aape1 ~]$ ls -la /mydata/
total 4
drwxr-xr-x.  2 awx  awx   41 Apr 28 09:27 .
dr-xr-xr-x. 19 root root 258 Apr 11 15:16 ..
-rw-r--r--.  1 awx  awx   33 Apr 11 12:34 file_read
-rw-r--r--.  1 awx  awx    0 Apr 28 09:27 file_write
```

簡単な Playbook を使って、アクセスを許可するために定義されたスリープでオートメーションを起動し、プロセスを理解して、ファイルの読み書きを示します。

```
# vim:ft=ansible:
```

```
- hosts: all
  gather_facts: false
  ignore_errors: yes
  vars:
    period: 120
    myfile: /mydata/file
  tasks:
    - name: Collect only selected facts
      ansible.builtin.setup:
        filter:
          - 'ansible_distribution'
          - 'ansible_machine_id'
          - 'ansible_memtotal_mb'
          - 'ansible_memfree_mb'
    - name: "I'm feeling real sleepy..."
      ansible.builtin.wait_for:
        timeout: "{{ period }}"
      delegate_to: localhost
    - ansible.builtin.debug:
        msg: "Isolated paths mounted into execution node: {{ AWX_ISOLATIONS_PATHS }}"
    - name: "Read pre-existing file..."
      ansible.builtin.debug:
        msg: "{{ lookup('file', '{{ myfile }}_read'
    - name: "Write to a new file..."
      ansible.builtin.copy:
        dest: "{{ myfile }}_write"
        content: |
          This is the file I've just written to.
```

```
    - name: "Read written out file..."
      ansible.builtin.debug:
        msg: "{{ lookup('file', '{{ myfile }}_write') }}"
```

Ansible Automation Platform 2 ナビゲーションパネルから **Settings** を選択します。次に、**Jobs** オプションから **Job settings** を選択します。

分離されたジョブを公開するパス:

```
[
"/mydata:/mydata:rw"
]
```

ボリュームマウントはコンテナー内で同じ名前でマップされ、読み取り/書き込み機能を備えています。これは、ジョブテンプレートを起動するときに使用されます。

実行ごとにスリープ期間を調整できるように、**起動時のプロンプト** を **extra_vars** に設定する必要があります。デフォルトは 30 秒です。

起動され、**wait_for** モジュールがスリープのために呼び出されたら、実行ノードに行き、何が実行されているかを確認できます。

実行が正常に完了したことを確認するには、次のコマンドを実行してジョブの出力を取得します。

```
$ podman exec -it 'podman ps -q' /bin/bash
bash-4.4#
```

これで、実行中の実行環境コンテナーに入りました。

権限を見ると、**awx** が root になっていることがわかりますが、ユーザーをサンドボックスに似たカーネル名前空間にマップするルートレス Podman を使用しているため、これはスーパーユーザーのような実際の root ではありません。shadow-utils 用のルートレス Podman の仕組みに関する詳細は、How does rootless Podman work? を参照してください。

```
bash-4.4# ls -la /mydata/
Total 4
drwxr-xr-x. 2 root root 41 Apr 28 09:27 .
dr-xr-xr-x. 1 root root 77 Apr 28 09:40 ..
-rw-r—--r–. 1 root root 33 Apr 11 12:34 file_read
-rw-r—--r–. 1 root root  0 Apr 28 09:27 file_write
```

結果を確認すると、このジョブは失敗しました。その理由を理解するには、残りの出力を調べる必要があります。

```
TASK [Read pre-existing file…]****************************** 10:50:12
ok: [localhost] => {
    “Msg”: “This is the file I am reading in.”

TASK {Write to a new file...}******************************** 10:50:12
An exception occurred during task execution. To see the full traceback, use -vvv. The error was:
PermissionError: [Errno 13] Permission denied: b'/mydata/.ansible_tmpazyqyqdrfile_write' -> b'
/mydata/file_write'
Fatal: [localhost]: FAILED! => {"changed": false, :checksum":
"9f576o85d584287a3516ee8b3385cc6f69bf9ce", "msg": "Unable to make
b'/root/.ansible/tmp/anisible-tim-1651139412.9808054-40-91081834383738/source' into
```

```
/mydata/file_write, failed final rename from b'/mydata/.ansible_tmpazyqyqdrfile_write': [Errno 13]
Permission denied: b'/mydata/.ansible_tmpazyqyqdrfile_write' -> b'/mydata/file_write}
...ignoring

TASK [Read written out file...] ****************************** 10:50:13
Fatal: [localhost]: FAILED: => {"msg": "An unhandled exception occurred while running the lookup
plugin 'file'. Error was a <class 'ansible.errors.AnsibleError;>, original message: could not locate file in
lookup: /mydate/file_write. Vould not locate file in lookup: /mydate/file_write"}
...ignoring
```

:rw が設定されているにもかかわらずジョブは失敗したため、書き込み機能が必要です。プロセスは既存のファイルを読み取ることはできましたが、書き出すことはできませんでした。これは、コンテナーにマウントされたボリュームコンテンツに適切なラベルを付ける必要がある SELinux 保護によるものです。ラベルがない場合、SELinux はコンテナー内でのプロセスの実行を阻止する可能性があります。OS によって設定されたラベルは、Podman によって変更されることはありません。詳細は、Podman ドキュメントを参照してください。

これはよくある誤解である可能性があります。デフォルトを **:z** に設定しました。これにより、Podman は共有ボリューム上のファイルオブジェクトのラベルを変更するように指示されます。

したがって、**:z** を追加することも、追加しないこともできます。

分離されたジョブを公開するパス:

```
[
   "/mydata:/mydata"
]
```

Playbook は期待どおりに機能するようになりました。

```
PLAY [all] ********************************************************* 11:05:52
TASK [I'm feeling real sleepy. . .] ************************** 11:05:52
ok: [localhost]
TASK [Read pre-existing file...] ***************************** 11:05:57
ok: [localhost] => {
 "Msg": "This is the file I'm reading in."
}
TASK [Write to a new file…] ********************************** 11:05:57
ok: [localhost]
TASK [Read written out file…] ******************************** 11:05:58
ok: [localhost] => {
      “Msg”: “This is the file I’ve just written to.”
```

基盤となる実行ノードのホストに戻ると、新しく書き出された内容があります。

注記

Red Hat OpenShift 内で自動化ジョブを起動するためにコンテナーグループを使用している場合、同じパスをその環境に公開するように Ansible Automation Platform 2 に指示することもできますが、設定でデフォルトを **On** に切り替える必要があります。

有効にすると、これは、実行に使用される Pod 仕様内に **volumeMounts** および **volumes** として注入します。以下に例を示します。

```
apiVersion: v1
kind: Pod
Spec:
   containers:
   - image: registry.redhat.io/ansible-automation-platform-24/ee-minimal-rhel8
  args:
    - ansible runner
    - worker
    - –private-data-dir=/runner
  volumeMounts:
mountPath: /mnt2
name: volume-0
readOnly: true
mountPath: /mnt3
name: volume-1
readOnly: true
mountPath: /mnt4
name: volume-2
readOnly: true
volumes:
hostPath:
  path: /mnt2
  type: “”
name: volume-0
hostPath:
  path: /mnt3
  type: “”
name: volume-1
hostPath:
  path: /mnt4
  type: “”
name: volume-2
```

実行中のコンテナー内のストレージはオーバーレイファイルシステムを使用しています。tmpfs がアンマウントされるのと同様に、実行中のコンテナー内の変更はジョブの完了後に破棄されます。