



Red Hat Ansible Automation Platform 2.3

Red Hat Ansible Automation Platform 自動化 メッシュガイド

このガイドでは、Ansible Automation Platform 環境の一部として自動化メッシュをデプロイするのに使用される情報を提供します。

Red Hat Ansible Automation Platform 2.3 Red Hat Ansible Automation Platform 自動化メッシュガイド

このガイドでは、Ansible Automation Platform 環境の一部として自動化メッシュをデプロイするのに使用される情報を提供します。

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[®] 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

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 RED HAT ANSIBLE AUTOMATION PLATFORM 環境での自動化メッシュのプランニング	5
1.1. 自動化メッシュについて	5
1.2. コントロールプレーンおよび実行プレーン	5
第2章 自動化メッシュの設定	9
2.1. 自動化メッシュのインストール	9
2.2. 認証局 (CA) 証明書のインポート	9
第3章 自動化メッシュの設計パターン	10
3.1. 複数ハイブリッドノードのインベントリーファイルの例	10
3.2. 単一の実行ノードを持つ単一ノードのコントロールプレーン	11
3.3. 回復力のある最小設定	13
3.4. 分離されたローカルおよびリモート実行設定	14
3.5. マルチホップ実行ノード	15
3.6. コントローラーノードへのアウトバウンドのみの接続	17
第4章 個々のノードまたはグループのプロビジョニング解除	19
4.1. インストーラーを使用した個別ノードのプロビジョニング解除	19
4.2. インストーラーを使用したグループのプロビジョニング解除	20

はじめに

Red Hat Ansible Automation Platform に興味をお持ちいただきありがとうございます。Ansible Automation Platform は、Ansible を装備した環境に、制御、ナレッジ、委譲の機能を追加して、チームが複雑かつ複数層のデプロイメントを管理できるように支援する商用サービスです。

このガイドでは、Ansible Automation Platform のインストールにおけるインストール要件およびプロセスについて説明します。このガイドの更新により、Ansible Automation Platform の最新リリースの情報が追加されました。

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、[弊社の CTO である Chris Wright のメッセージ](#) を参照してください。

第1章 RED HAT ANSIBLE AUTOMATION PLATFORM 環境での自動化メッシュのプランニング

以下のトピックでは、Ansible Automation Platform 環境で自動化メッシュのデプロイメントを計画するのに役立つ情報を提供します。以降のセクションでは、自動化メッシュトポロジを設計する方法の例に加えて、自動化メッシュを構成する概念を説明します。シンプルなトポロジから複雑なトポロジまでを例に使用して、自動化メッシュをデプロイするための各種の方法を示します。

1.1. 自動化メッシュについて

自動化メッシュは、既存ネットワークを使用して互いにピアツーピア接続を確立しているノードを介して、大規模な分散ワーカーのコレクション全体で作業分散を容易にするオーバーレイネットワークです。

Red Hat Ansible Automation Platform 2 は、Ansible Tower と分離されたノードを自動化コントローラーおよび自動化ハブに置き換えます。自動化コントローラーは、UI、Restful API、RBAC、ワークフローおよび CI/CD 統合を介して自動化のコントロールプレーンを提供します。一方、Automation Mesh は、制御および実行レイヤーを構成するノードの設定、検出、変更、または修正に使用できます。

Automation Mesh では、以下が追加されました。

- 個別にスケーリングする動的クラスター容量。これにより、ダウンタイムを最小限に抑えてノードを作成、登録、グループ化、グループ化解除、および登録解除できます。
- コントロールプレーンと実行プレーンの分離。コントロールプレーンの容量とは関係なく Playbook の実行容量をスケーリングできます。
- レイテンシーに対して回復性があり、サービスを停止させずに設定変更が可能で、またサービスの停止が存在する場合に動的に再ルーティングを行い別のパスを選択するデプロイメントの選択。メッシュルーティングの変更。
- FIPS (Federal Information Processing Standards) に準拠する双方向、マルチホップのメッシュ通信の可能性を含む接続性。

1.2. コントロールプレーンおよび実行プレーン

自動化メッシュでは、ユニークなノードタイプを使用して **コントロール プレーン** と **実行 プレーン** の両方を作成します。自動化メッシュトポロジを設計する前に、コントロールプレーンおよび実行プレーン、ならびにそのノードタイプの詳細を確認してください。

1.2.1. コントロールプレーン

コントロールプレーン は、ハイブリッドノードおよびコントロールノードで構成されます。コントロールプレーンのインスタンスは、プロジェクト更新や管理ジョブに加えて、Web サーバーやタスクディスパッチャーなどの永続的な自動化コントローラーサービスを実行します。

- **ハイブリッドノード**: これはコントロールプレーンノードのデフォルトのノードタイプです。これは、プロジェクト更新、管理ジョブ、**ansible-runner** タスク操作などの自動化コントローラーランタイム機能に対応します。ハイブリッドノードは、自動化の実行にも使用されます。
- **コントロールノード**: コントロールノードは、プロジェクトおよびインベントリを更新およびシステムジョブを実行しますが、通常のジョブは実行しません。これらのノードでは実行機能は無効にされます。

1.2.2. 実行プレーン

実行プレーン は、コントロールプレーンの代わりに自動化を実行し制御機能を持たない実行ノードで構成されます。ホップノードは、通信に対応します。**実行プレーン** のノードは、地理的にコントロールプレーンから離れた、レイテンシーの高いユーザー空間のジョブのみを実行します。

- **実行ノード** - 実行ノードは、**podman** 分離により **ansible-runner** でジョブを実行します。このノードタイプは分離されたノードに似ています。これは、実行プレーンノードのデフォルトのノードタイプです。
- **ホップノード** - ジャンプホストと同様に、ホップノードはトラフィックを他の実行ノードにルーティングします。ホップノードは自動化を実行できません。

1.2.3. ピア

ピアの関係はノード間の接続を定義します。**[automationcontroller]** グループおよび **[execution_nodes]** グループ内にピアを定義するか、**[automationcontroller:vars]** または **[execution_nodes:vars]** グループを使用します。

1.2.4. 自動化メッシュノードタイプの定義

このセクションの例は、インベントリーファイルでホストのノードタイプを設定する方法を示しています。

コントロールプレーンまたは実行プレーンインベントリーグループ内の単一ノードの **node_type** を設定できます。ノードのグループ全体のノードタイプを定義するには、グループの **vars** スタンザで **node_type** を設定します。

- コントロールプレーン **automationcontroller** グループの **node_type** に許可される値は、**Hybrid** (デフォルト) と **control** です。
- **[execution_nodes]** グループの **node_type** に許可される値は、**execution** (デフォルト) と **hop** です。

ハイブリッドノード

次のインベントリーは、コントロールプレーンの単一のハイブリッドノードで設定されています。

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

コントロールノード

次のインベントリーは、コントロールプレーンの単一のコントロールノードで設定されています。

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

コントロールプレーンノードの **vars** スタンザで **node_type** を **control** に設定すると、コントロールプレーン内のすべてのノードがコントロールノードになります。

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

```
[automationcontroller:vars]
node_type=control
```

実行ノード

次のスタanzasは、実行プレーンで単一の実行ノードを定義します。

```
[execution_nodes]
execution-plane-1.example.com
```

ホップノード

次のスタanzasは、実行プレーン内の単一のホップノードと実行ノードを定義します。**node_type** 変数は、個々のノードごとに設定されます。

```
[execution_nodes]
execution-plane-1.example.com node_type=hop
execution-plane-2.example.com
```

node-type をグループレベルで設定する場合は、実行ノードとホップノードに個別のグループを作成する必要があります。

```
[execution_nodes]
execution-plane-1.example.com
execution-plane-2.example.com

[execution_group]
execution-plane-2.example.com

[execution_group:vars]
node_type=execution

[hop_group]
execution-plane-1.example.com

[hop_group:vars]
node_type=hop
```

ピア接続

peers= ホスト変数を使用して、ノード間の接続を作成します。以下の例では、**control-plane-1.example.com** を **execution-node-1.example.com** に、**execution-node-1.example.com** を **execution-node-2.example.com** に、それぞれ接続します。

```
[automationcontroller]
control-plane-1.example.com peers=execution-node-1.example.com

[automationcontroller:vars]
node_type=control

[execution_nodes]
execution-node-1.example.com peers=execution-node-2.example.com
execution-node-2.example.com
```

関連情報

- メッシュノードの実装方法の例については、このガイドの自動化メッシュトポロジーの例を参照してください。

第2章 自動化メッシュの設定

Ansible Automation Platform インストーラーを設定して、Ansible 環境の自動化メッシュをセットアップします。認証局 (CA) 証明書のインポートなど、インストールをカスタマイズするための追加のタスクを実行します。

2.1. 自動化メッシュのインストール

Ansible Automation Platform インストールプログラムを使用して、自動化メッシュをセットアップするか、自動化メッシュにアップグレードします。メッシュネットワーク内のノード、グループ、およびピア関係に関する詳細を Ansible Automation Platform に提供する場合は、インストーラーバンドルの **inventory** ファイルで定義します。

関連情報

- [Red Hat Ansible Automation Platform インストールガイド](#)
- [Automation Mesh Design Patterns](#)

2.2. 認証局 (CA) 証明書のインポート

認証局 (CA) は、自動化メッシュ環境で個々のノード証明書を検証して署名します。Red Hat Ansible Automation Platform インストーラーの **inventory** ファイルで証明書のパスと秘密 RSA キーファイルを指定すると、独自の CA を提供できます。



注記

CA を提供しないと、Ansible Automation Platform インストールプログラムが CA を生成します。

手順

1. 編集する **inventory** ファイルを開きます。
2. **mesh_ca_keyfile** 変数を追加し、秘密 RSA キー (**.key**) へのフルパスを指定します。
3. **mesh_ca_certfile** 変数を追加し、CA 証明書ファイル (**.crt**) へのフルパスを指定します。
4. 変更をインベントリーファイルに保存します。

例

```
[all:vars]
mesh_ca_keyfile=/tmp/<mesh_CA>.key
mesh_ca_certfile=/tmp/<mesh_CA>.crt
```

インベントリーファイルに CA ファイルを追加したら、インストールプログラムを実行して CA を適用します。このプロセスは、CA をメッシュネットワーク上の各コントロールノードおよび実行ノードの **/etc/receptor/tls/ca/** ディレクトリーにコピーします。

関連情報

- [Red Hat Ansible Automation Platform System Requirements](#)

第3章 自動化メッシュの設計パターン

このセクションの自動化メッシュトポロジは、環境でメッシュのデプロイメントを設計するのに使用できる例を提供しています。単純なハイブリッドノードのデプロイメントから、複数の自動化コントローラーインスタンスをデプロイし、複数の実行ノードおよびホップノードを使用する複雑なパターンまで、さまざまな例が用意されています。

前提条件

- ノードタイプと関係に関する概念的な情報を確認している。



注記

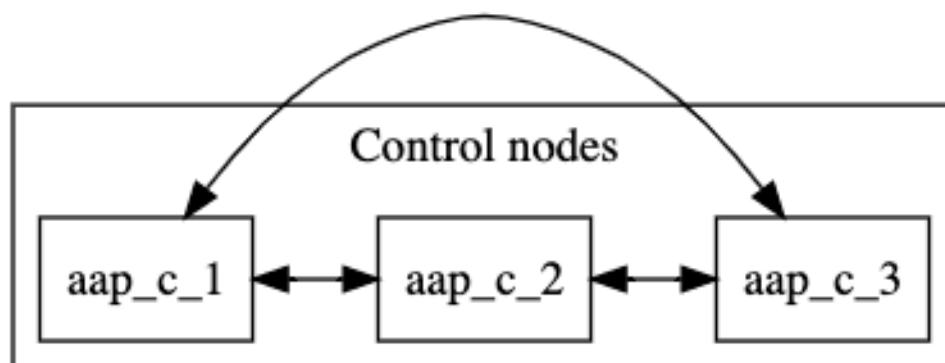
次の例には、メッシュトポロジを示すイメージが含まれています。イメージの矢印は、ピアリングの方向を示しています。ピアリングが確立されると、ノード間の接続により双方向通信が可能になります。

3.1. 複数ハイブリッドノードのインベントリーファイルの例

このインベントリーファイルのサンプルは、複数のハイブリッドノードで構成されるコントロールプレーンをデプロイします。コントロールプレーンのノードは、自動的に相互にピアリングされます。

```
[automationcontroller]
aap_c_1.example.com
aap_c_2.example.com
aap_c_3.example.com
```

次の図は、このメッシュネットワークのトポロジを示しています。



コントロールプレーンのノードのデフォルトの **node_type** は **Hybrid** です。[automationcontroller group] で個々のノードの **node_type** を **hybrid** に明示的に設定できます。

```
[automationcontroller]
aap_c_1.example.com node_type=hybrid
aap_c_2.example.com node_type=hybrid
aap_c_3.example.com node_type=hybrid
```

または、[automationcontroller] グループ内の全ノードの **node-type** を設定できます。コントロールプレーンに新しいノードを追加すると、自動的にハイブリッドノードに設定されます。

```
[automationcontroller]
```

```
aap_c_1.example.com
aap_c_2.example.com
aap_c_3.example.com

[automationcontroller:vars]
node_type=hybrid
```

将来、コントロールプレーンにコントロールノードを追加する可能性がある場合は、ハイブリッドノード用に別のグループを定義し、グループの **node-type** を設定することが推奨されます。

```
[automationcontroller]
aap_c_1.example.com
aap_c_2.example.com
aap_c_3.example.com

[hybrid_group]
aap_c_1.example.com
aap_c_2.example.com
aap_c_3.example.com

[hybrid_group:vars]
node_type=hybrid
```

3.2. 単一の実行ノードを持つ単一ノードのコントロールプレーン

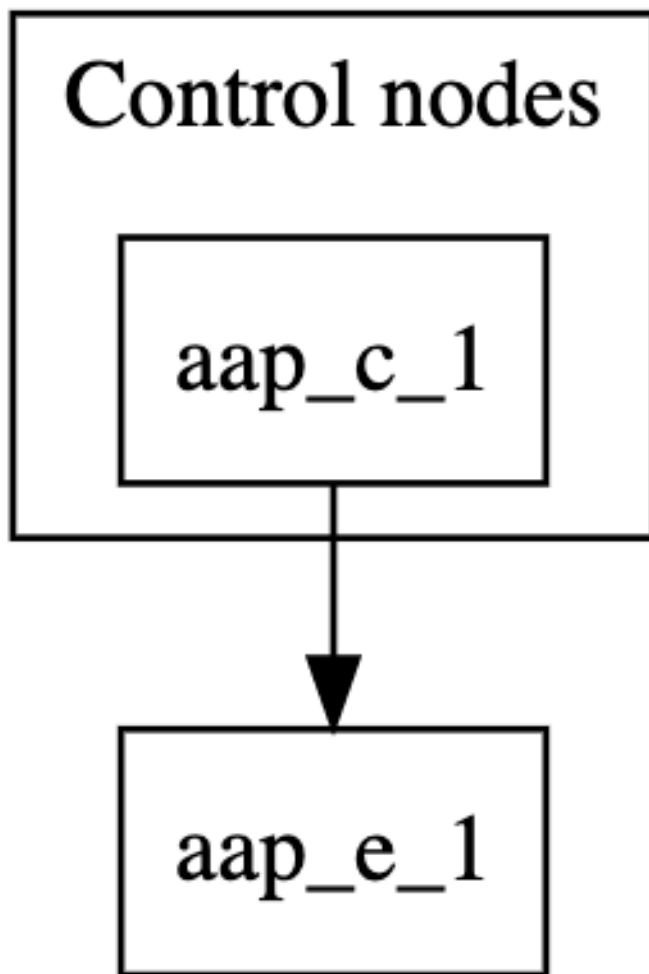
このサンプルインベントリーファイルは、単一ノードのコントロールプレーンをデプロイし、実行ノードに対してピア関係を確立します。

```
[automationcontroller]
aap_c_1.example.com

[automationcontroller:vars]
node_type=control
peers=execution_nodes

[execution_nodes]
aap_e_1.example.com
```

次の図は、このメッシュネットワークのトポロジーを示しています。



[automationcontroller] スタンザは制御ノードを定義します。自動化コントローラグループに新しいノードを追加すると、自動的に **aap_c_1.example.com** ノードとピアリングされます。

[automationcontroller:vars] スタンザは、ノードタイプを **control** プレーン内のすべてのノードを制御するように設定し、ノードが実行ノードとどのようにピアリングするかを定義します。

- **execution_nodes** グループに新しいノードを追加すると、コントロールプレーンノードは自動的にそれにピアリングします。
- 新しいノードを **AutomationController** グループに追加すると、ノードタイプは **control** に設定されます。

[execution_nodes] スタンザは、インベントリ内の全実行ノードとホップノードを一覧表示します。デフォルトのノードタイプは **execution** です。個々のノードのノードタイプを指定できます。

```
[execution_nodes]
aap_e_1.example.com node_type=execution
```

または、**[execution_nodes]** グループ内の全実行ノードの **node_type** を設定できます。グループに新しいノードを追加すると、そのノードは自動的に実行ノードに設定されます。

```
[execution_nodes]
aap_e_1.example.com
```



```
[execution_nodes:vars]
node_type=execution
```

将来ホップノードをインベントリーに追加する予定がある場合は、実行ノード用に別のグループを定義し、グループの **node_type** を設定することが推奨されます。

```
[execution_nodes]
aap_e_1.example.com

[local_execution_group]
aap_e_1.example.com

[local_execution_group:vars]
node_type=execution
```

3.3. 回復力のある最小設定

このサンプルインベントリーファイルは、2つのコントロールノードと2つの実行ノードで設定されるコントロールプレーンをデプロイします。コントロールプレーン内のすべてのノードは、自動的に相互にピアリングされます。コントロールプレーンのすべてのノードは、**execution_nodes** グループのすべてのノードとピアリングされます。この設定は、すべての制御ノードから実行ノードに到達できるため、回復力があります。

容量アルゴリズムは、ジョブが開始されたときに選択される制御ノードを決定します。詳細は、**Automation Controller ユーザーガイド**の [Automation controller Capacity Determination and Job Impact](#) を参照してください。

次のインベントリーファイルは、この設定を定義します。

```
[automationcontroller]
aap_c_1.example.com
aap_c_2.example.com

[automationcontroller:vars]
node_type=control
peers=execution_nodes

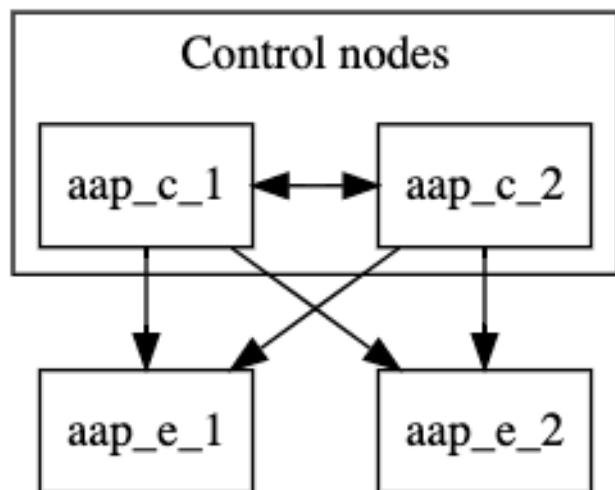
[execution_nodes]
aap_e_1.example.com
aap_e_1.example.com
```

[automationcontroller] スタンザは制御ノードを定義します。コントロールプレーン内のすべてのノードは、相互にピアリングされます。新しいノードを **automationcontroller** グループに追加すると、元のノードと自動的にピアリングされます。

[automationcontroller:vars] スタンザは、ノードタイプを **control** プレーン内のすべてのノードを制御するように設定し、ノードが実行ノードとどのようにピアリングするかを定義します。

- **execution_nodes** グループに新しいノードを追加すると、コントロールプレーンノードは自動的にそれにピアリングします。
- 新しいノードを **AutomationController** グループに追加すると、ノードタイプは **control** に設定されます。

次の図は、このメッシュネットワークのトポロジーを示しています。



3.4. 分離されたローカルおよびリモート実行設定

この設定は、ホップノードとリモート実行ノードを回復力のある設定に追加します。リモート実行ノードは、ホップノードから到達可能です。

リモートの場所に実行ノードをセットアップする場合、または DMZ ネットワークで自動化を実行する必要がある場合は、このセットアップを使用できます。

```

[automationcontroller]
aap_c_1.example.com
aap_c_2.example.com

[automationcontroller:vars]
node_type=control
peers=instance_group_local

[execution_nodes]
aap_e_1.example.com
aap_e_2.example.com
aap_h_1.example.com
aap_e_3.example.com

[instance_group_local]
aap_e_1.example.com
aap_e_2.example.com

[hop]
aap_h_1.example.com

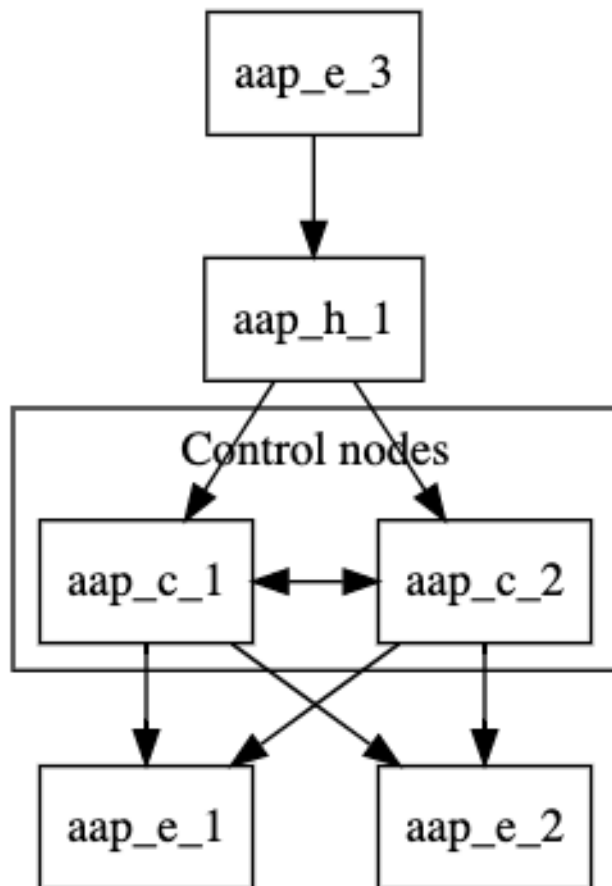
[hop:vars]
peers=automationcontroller

[instance_group_remote]
aap_e_3.example.com

[instance_group_remote:vars]
peers=hop

```

次の図は、このメッシュネットワークのトポロジを示しています。



[automationcontroller:vars] スタンザは、コントロールプレーン内の全ノードのノードタイプを設定し、コントロールノードがローカル実行ノードとピアリングする方法を定義します。

- コントロールプレーン内のすべてのノードは、自動的に相互にピアリングされます。
- コントロールプレーンのすべてのノードは、すべてのローカル実行ノードとピアリングされます。

ノードのグループの名前が **instance_group_** で始まる場合、インストーラーはそれをインスタンスグループとして認識し、Ansible Automation Platform ユーザーインターフェイスに追加します。

3.5. マルチホップ実行ノード

この設定では、回復力のあるコントローラーノードが回復力のあるローカル実行ノードとピアリングされます。回復力のあるローカルホップノードは、コントローラーノードとピアリングされます。リモート実行ノードとリモートホップノードは、ローカルホップノードとピアリングされます。

リモートネットワークから DMZ ネットワークで自動化を実行する必要がある場合は、このセットアップを使用できます。

```

[automationcontroller]
aap_c_1.example.com
aap_c_2.example.com
aap_c_3.example.com

[automationcontroller:vars]
node_type=control
peers=instance_group_local
  
```

```
[execution_nodes]
aap_e_1.example.com
aap_e_2.example.com
aap_e_3.example.com
aap_e_4.example.com
aap_h_1.example.com node_type=hop
aap_h_2.example.com node_type=hop
aap_h_3.example.com node_type=hop

[instance_group_local]
aap_e_1.example.com
aap_e_2.example.com

[instance_group_remote]
aap_e_3.example.com

[instance_group_remote:vars]
peers=local_hop

[instance_group_multi_hop_remote]
aap_e_4.example.com

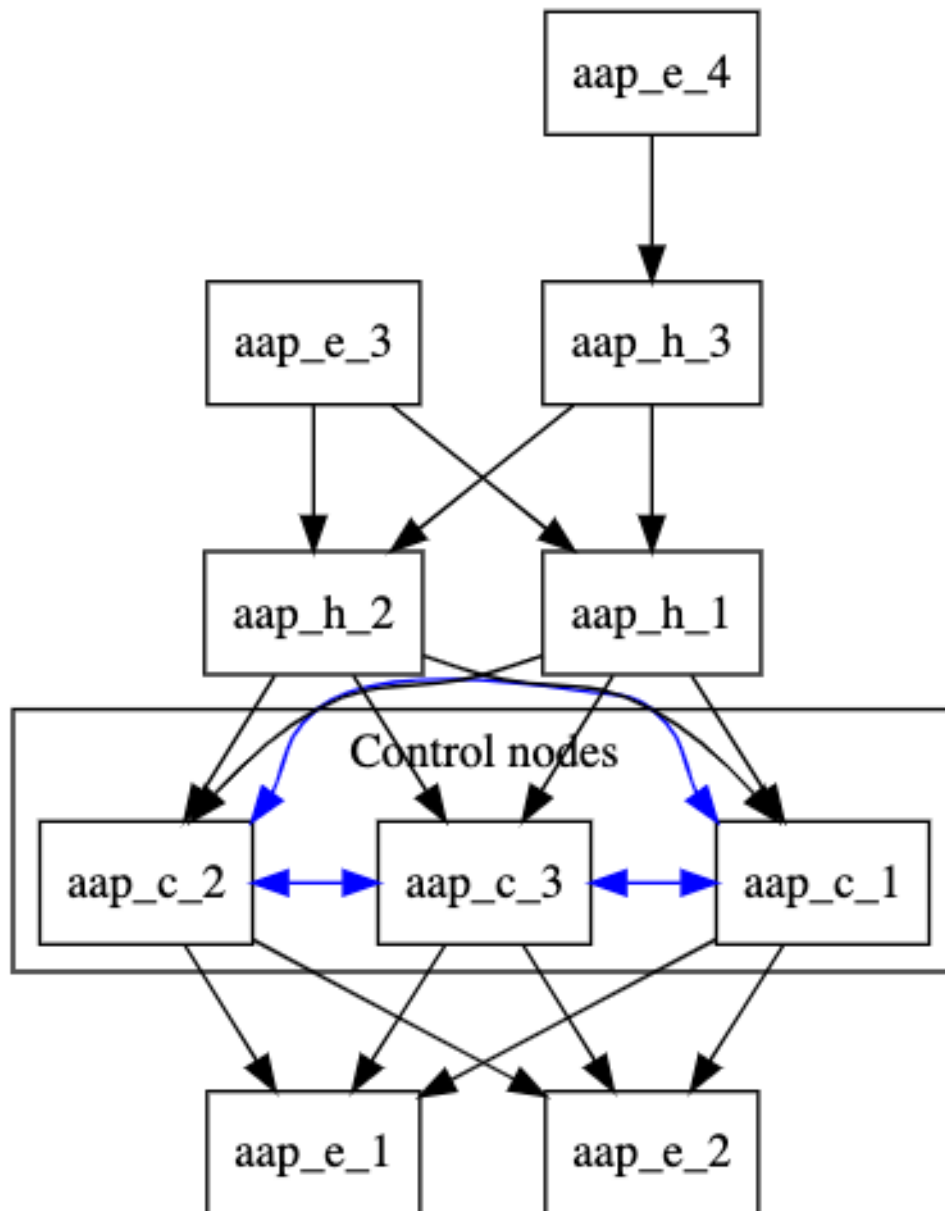
[instance_group_multi_hop_remote:vars]
peers=remote_multi_hop

[local_hop]
aap_h_1.example.com
aap_h_2.example.com

[local_hop:vars]
peers=automationcontroller

[remote_multi_hop]
aap_h_3 peers=local_hop
```

次の図は、このメッシュネットワークのトポロジーを示しています。



[automationcontroller:vars] スタンザは、コントロールプレーン内の全ノードのノードタイプを設定し、コントロールノードがローカル実行ノードとピアリングする方法を定義します。

- コントロールプレーン内のすべてのノードは、自動的に相互にピアリングされます。
- コントロールプレーンのすべてのノードは、すべてのローカル実行ノードとピアリングされます。

[local_hop:vars] スタンザは、**local_hop** グループ内のすべてのノードをすべての制御ノードとピアリングします。

ノードのグループの名前が **instance_group_** で始まる場合、インストーラーはそれをインスタンスグループとして認識し、Ansible Automation Platform ユーザーインターフェイスに追加します。

3.6. コントローラーノードへのアウトバウンドのみの接続

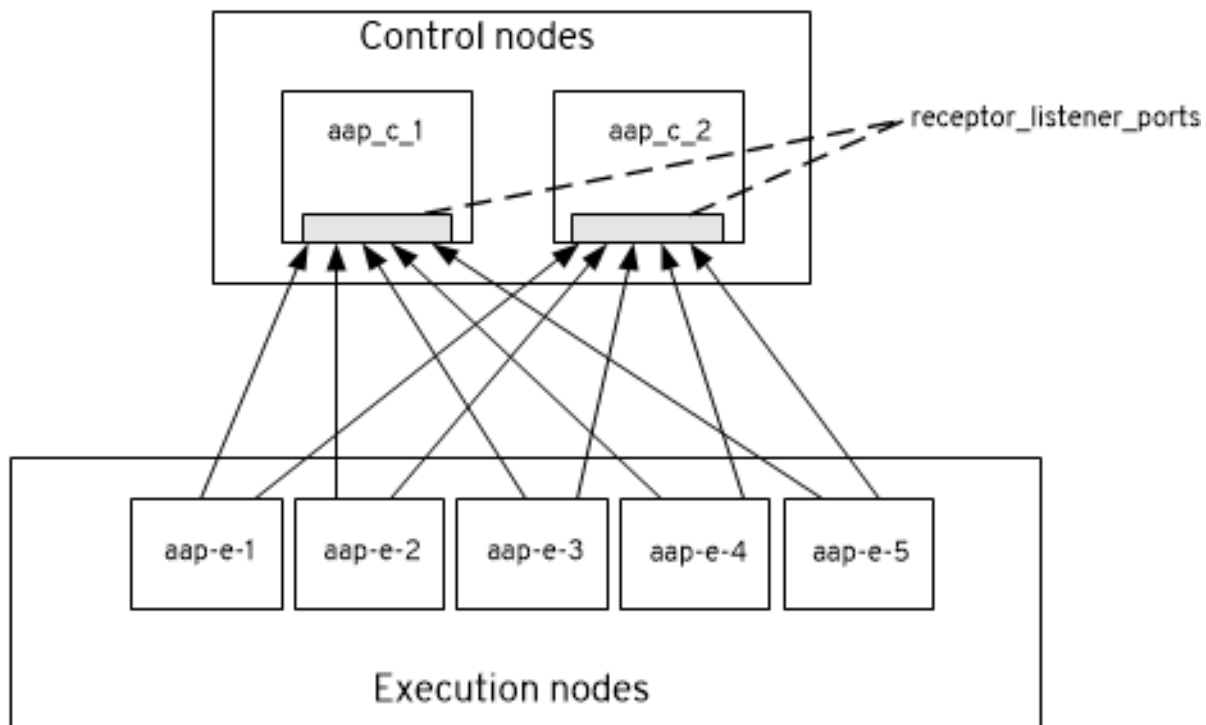
このサンプルインベントリーファイルは、複数のコントロールノードと2つの実行ノードで設定されるコントロールプレーンをデプロイします。コントローラーノードへの送信接続のみが許可されます。**execution_nodes** グループ内のすべてのノードは、コントローラープレーン内のすべてのノードとピアリングされます。

```
[automationcontroller]  
controller-[1:2].example.com
```

```
[execution_nodes]  
execution-[1:5].example.com
```

```
[execution_nodes:vars]  
# connection is established *from* the execution nodes *to* the automationcontroller  
peers=automationcontroller
```

次の図は、このメッシュネットワークのトポロジーを示しています。



第4章 個々のノードまたはグループのプロビジョニング解除

Ansible Automation Platform インストーラーを使用して、自動化メッシュノードおよびインスタンスグループのプロビジョニングを解除できます。本セクションの手順では、インベントリーファイルのサンプルを使用して、特定のノードまたはグループ全体のプロビジョニングを解除する方法を説明します。

4.1. インストーラーを使用した個別ノードのプロビジョニング解除

Ansible Automation Platform インストーラーを使用して、自動化メッシュからノードのプロビジョニングを解除できます。**inventory** ファイルを編集して、プロビジョニングを解除するノードをマークしてから、インストーラーを実行します。インストーラーを実行すると、ノードに割り当てられたすべての設定ファイルおよびログも削除されます。



注記

[automationcontroller] グループで指定されている最初のホストを除き、インベントリーの任意のホストのプロビジョニングを解除することができます。

手順

- インストーラーファイルで、プロビジョニング解除するノードに **node_state=deprovision** を追加します。

例

このインベントリーファイルのサンプルでは、自動化メッシュ設定から2つのノードがプロビジョニング解除されます。

```
[automationcontroller]
126-addr.tatu.home ansible_host=192.168.111.126 node_type=control
121-addr.tatu.home ansible_host=192.168.111.121 node_type=hybrid routable_hostname=121-addr.tatu.home
115-addr.tatu.home ansible_host=192.168.111.115 node_type=hybrid node_state=deprovision

[automationcontroller:vars]
peers=connected_nodes

[execution_nodes]
110-addr.tatu.home ansible_host=192.168.111.110 receptor_listener_port=8928
108-addr.tatu.home ansible_host=192.168.111.108 receptor_listener_port=29182
node_state=deprovision
100-addr.tatu.home ansible_host=192.168.111.100 peers=110-addr.tatu.home node_type=hop
```

4.1.1. 分離されたノードのプロビジョニング解除

awx-manage プロビジョニング解除ユーティリティを使用して、分離されたノードを手動で削除するオプションがあります。



警告

プロビジョニング解除コマンドを使用して、実行ノードに移行されていない分離ノードのみを削除します。自動化メッシュアーキテクチャーから実行ノードのプロビジョニングを解除するには、代わりに [インストーラーメソッド](#) を使用します。

手順

1. インスタンスをシャットダウンします。

```
$ automation-controller-service stop
```

2. 別のインスタンスからプロビジョニング解除コマンドを実行し、**host_name** をインベントリーファイルにリストされているノードの名前に置き換えます。

```
$ awx-manage deprovision_instance --hostname=<host_name>
```

4.2. インストーラーを使用したグループのプロビジョニング解除

Ansible Automation Platform インストーラーを使用して、自動化メッシュからグループ全体のプロビジョニングを解除できます。インストーラーを実行すると、グループ内のノードに割り当てられたすべての設定ファイルおよびログが削除されます。



注記

[automationcontroller] グループで指定されている最初のホストを除き、インベントリーの任意のホストのプロビジョニングを解除することができます。

手順

- **node_state=deprovision** を、プロビジョニング解除するグループに関連付けられた [group:vars] に追加します。

例

```
[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
execution-node-5.example.com peers=execution-node-6.example.com
execution-node-6.example.com peers=execution-node-7.example.com
execution-node-7.example.com
```

```
[execution_nodes:vars]
node_state=deprovision
```

4.2.1. 分離されたインスタンスグループのプロビジョニング解除

awx-manage プロビジョニング解除ユーティリティーを使用して、分離されたインスタンスグループを手動で削除するオプションがあります。



警告

分離されたインスタンスグループのみを削除するには、プロビジョニング解除コマンドを使用します。自動化メッシュアーキテクチャーからインスタンスグループのプロビジョニングを解除するには、代わりに [インストーラーメソッド](#) を使用します。

手順

- 次のコマンドを実行し、**<name>** をインスタンスグループの名前に置き換えます。

```
$ awx-manage unregister_queue --queueName=<name>
```