Red Hat Ansible Automation Platform 2.4

VM 기반 설치를 위한 Red Hat Ansible Automation Platform Automation Mesh 가이드

클라우드 네이티브 방식으로 대규모로 자동화

Last Updated: 2024-06-13

Red Hat Ansible Automation Platform 2.4 VM 기반 설치를 위한 Red Hat Ansible Automation Platform Automation Mesh 가이드

클라우드 네이티브 방식으로 대규모로 자동화

법적 공지

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.

초록

이 가이드에서는 VM 기반 Ansible Automation Platform 환경의 일부로 자동화 메시를 배포하는 방법을 보여줍니다.

차례

머리말

Red Hat Ansible Automation Platform에 관심을 가져 주셔서 감사합니다. Ansible Automation Platform은 Ansible 기반 환경에 제어, 지식, 위임을 추가하여 팀이 복잡한 다중 계층 배포를 관리하는 데 도움이 되는 상용 서비스입니다.

이 가이드에서는 Red Hat Ansible Automation Platform의 VM 기반 설치에서 자동화 메시를 설정하는 데 필요한 요구 사항과 프로세스를 이해하는 데 도움이 됩니다. 이 문서는 최신 Ansible Automation Platform 릴리스에 대한 정보를 포함하도록 업데이트되었습니다.

RED HAT 문서에 관한 피드백 제공

이 문서를 개선하기 위한 제안이 있거나 오류를 찾을 수 있는 경우 https://access.redhat.com 에서 기술 지원에 문의하여 **docs-product** 구성 요소를 사용하여 Ansible Automation Platform Jira 프로젝트에 문제를 생성하십시오.

1장. VM 기반 RED HAT ANSIBLE AUTOMATION PLATFORM 환경에서 자동화 메시 계획

다음 주제에는 VM 기반 Ansible Automation Platform 환경에서 자동화 메시 배포를 계획하는 데 도움이 되는 정보가 포함되어 있습니다. 다음 섹션에서는 자동화 메시 토폴로지를 설계할 수 있는 방법에 대한 예제 외에 자동화 메시를 구성하는 개념에 대해 설명합니다. 자동화 메시를 배포할 수 있는 다양한 방법을 설명하기 위해 복잡한 토폴로지 예제가 포함되어 있습니다.

1.1. 자동화 메시 정보

자동화 메시는 기존 네트워크를 사용하여 서로 피어 투 피어 연결을 설정하는 노드를 통해 대규모 작업자 컬렉션에서 작업을 쉽게 배포하기 위한 오버레이 네트워크입니다.

Red Hat Ansible Automation Platform 2는 Ansible Tower 및 격리된 노드를 자동화 컨트롤러 및 자동화 허브로 대체합니다. 자동화 컨트롤러는 UI, RESTful API, RBAC, 워크플로 및 CI/CD 통합을 통해 자동화할 컨트롤 플레인을 제공하는 반면 자동화 메시를 사용하여 제어 및 실행 계층을 형성하는 노드를 설정, 검색, 변경 또는 수정할 수 있습니다.

자동화 메시는 통신에 TLS 암호화를 사용하므로 외부 네트워크(인터넷 또는 기타)를 통과하는 트래픽이 전송 중에 암호화됩니다.

자동화 메시의 도입:

- 독립적으로 확장되는 동적 클러스터 용량을 사용하면 다운타임을 최소화하여 노드를 생성, 등록, 그룹 해제 및 등록할 수 있습니다.
- 컨트롤 플레인 용량과 독립적으로 플레이북 실행 용량을 확장할 수 있는 컨트롤 및 실행 플레인 분리입니다.
- 대기 시간에 탄력적이고, 중단 없이 재구성할 수 있는 배포 선택, 중단 없이 재구성할 수 있으며, 중단 시 다른 경로를 동적으로 다시 라우팅할 수 있습니다.
- FIPS(*Federal Information Processing Standards*)와 호환되는 양방향 다중 홉 메시 통신 가능성이 포함된 연결입니다.

1.2. 컨트롤 및 실행 플레인

자동화 메시는 고유한 노드 유형을 사용하여 **컨트롤** 및 **실행** 플레인을 모두 생성합니다. 자동화 메시 토폴로지를 설계하기 전에 컨트롤 및 실행 플레인 및 해당 노드 유형에 대해 자세히 알아보십시오.

1.2.1. 컨트롤 플레인

컨트롤 플레인 은 하이브리드 노드 및 컨트롤 노드로 구성됩니다. 컨트롤 플레인의 인스턴스는 프로젝트 업데이트 및 관리 작업 외에도 웹 서버 및 작업 디스패처와 같은 영구 자동화 컨트롤러 서비스를 실행합니다.

- **하이브리드 노드** - 프로젝트 업데이트, 관리 작업 및 **ansible-runner** 작업 작업과 같은 자동화 컨트롤러 런타임 기능을 담당하는 컨트롤 플레인 노드의 기본 노드 유형입니다. 하이브리드 노드는 자동화 실행에도 사용됩니다.
- **컨트롤 노드** - 컨트롤 노드는 일반 작업이 아닌 프로젝트 및 인벤토리 업데이트 및 시스템 작업을 실행합니다. 이러한 노드에서 실행 기능이 비활성화되어 있습니다.

1.2.2. 실행 플레인

실행 플레인은 컨트롤 플레인 을 대신하여 자동화를 실행하고 컨트롤 기능이 없는 실행 노드로 구성됩니다. 홉 노드는 통신할 수 있습니다. **실행 플레인** 의 노드는 사용자 공간 작업만 실행하며 컨트롤 플레인에서 대기 시간이 긴 지리적으로 구분될 수 있습니다.

- **실행 노드** - 실행 노드는 **podman** 격리를 사용하여 **ansible-runner** 에서 작업을 실행합니다. 이 노드 유형은 격리된 노드와 유사합니다. 이는 실행 플레인 노드의 기본 노드 유형입니다.
- **홉 노드** - 점프 호스트와 유사하게 홉 노드는 트래픽을 다른 실행 노드로 라우팅합니다. 홉 노드는 자동화를 실행할 수 없습니다.

1.2.3. 피어

피어 관계는 노드 간 연결을 정의합니다. **[automationcontroller]** 및 **[execution_nodes]** 그룹 또는 **[automationcontroller:vars]** 또는 **[execution_nodes:vars]** 그룹을 사용하여 피어를 정의할 수 있습니다.

1.2.4. 자동화 메시 노드 유형 정의

이 섹션의 예제에서는 인벤토리 파일의 호스트에 대한 노드 유형을 설정하는 방법을 보여줍니다.

컨트롤 플레인 또는 실행 플레인 인벤토리 그룹에서 단일 노드의 **node_type** 을 설정할 수 있습니다. 전체 노드 그룹의 노드 유형을 정의하려면 그룹의 **vars** 스탠자에 **node_type** 을 설정합니다.

- 컨트롤 플레인 **[automationcontroller]** 그룹에서 **node_type** 에 허용되는 값은 **하이브리드** (기본값) 및 **제어** 입니다.
- **[execution_nodes]** 그룹에서 **node_type** 에 허용되는 값은 **execution** (기본값) 및 **홉** 입니다.

하이브리드 노드

다음 인벤토리는 컨트롤 플레인의 단일 하이브리드 노드로 구성됩니다.

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

제어 노드

다음 인벤토리는 컨트롤 플레인의 단일 컨트롤 노드로 구성됩니다.

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

컨트롤 플레인 노드의 **vars** 스탠자에서 **node_type** 을 **제어** 하도록 설정하면 컨트롤 플레인의 모든 노드가 컨트롤 노드입니다.

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

[automationcontroller:vars]
node_type=control
```

실행 노드

다음 스탠자는 실행 플레인에서 단일 실행 노드를 정의합니다.

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

홉 노드

다음 스탠자는 실행 플레인에서 단일 홉 노드와 실행 노드를 정의합니다. **node_type** 변수는 모든 개별 노드에 대해 설정됩니다.

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

그룹 수준에서 **node_type** 을 설정하려면 실행 노드 및 홉 노드에 대해 별도의 그룹을 생성해야 합니다.

```
[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
```

피어 연결

peers= 호스트 변수를 사용하여 노드 간 연결을 생성합니다. 다음 예제에서는 **control-plane-1.example.com** 을 **execution-node-1.example.com** 에 연결하고 **execution-node-1.example.com** 을 **execution-node-2.example.com** 에 연결합니다.

```
[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
```

추가 리소스

- 메시 노드를 구현하는 방법에 대한 자세한 내용은 이 가이드의 자동화 메시 토폴로지 예제를 참조하십시오.

2장. 자동화 메시 설정

Ansible Automation Platform 설치 프로그램을 구성하여 Ansible 환경에 대한 자동화 메시를 설정합니다. CA(인증 기관) 인증서 가져오기와 같은 설치를 사용자 지정하여 추가 작업을 수행합니다.

2.1. 자동화 메시 설치

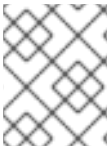
Ansible Automation Platform 설치 프로그램을 사용하여 자동화 메시를 설정하거나 자동화 메시로 업그레이드합니다. 메시 네트워크의 노드, 그룹 및 피어 관계에 대한 세부 정보를 Ansible Automation Platform에 제공하려면 설치 프로그램 번들의 **인벤토리** 파일에 해당 관계를 정의합니다.

추가 리소스

- Red Hat Ansible Automation Platform 설치 가이드
- 자동화 메시 설계 패턴

2.2. CA(인증 기관) 인증서 가져오기

CA(인증 기관)는 자동화 메시 환경에서 개별 노드 인증서를 확인하고 서명합니다. Red Hat Ansible Automation Platform 설치 프로그램의 **인벤토리** 파일에 인증서의 경로와 개인 RSA 키 파일을 지정하여 자체 CA를 제공할 수 있습니다.



참고

Ansible Automation Platform 설치 프로그램은 CA를 제공하지 않는 경우 CA를 생성합니다.

프로세스

1. 편집할 **인벤토리** 파일을 엽니다.
2. **mesh_ca_keyfile** 변수를 추가하고 개인 RSA 키(**.key**)의 전체 경로를 지정합니다.
3. **mesh_ca_certfile** 변수를 추가하고 CA 인증서 파일의 전체 경로(**.crt**)를 지정합니다.
4. 인벤토리 파일에 변경 사항을 저장합니다.

예제

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

인벤토리 파일에 CA 파일이 추가되면 설치 프로그램을 실행하여 CA를 적용합니다. 이 프로세스는 메시 네트워크의 각 제어 및 실행 노드의 **/etc/receptor/tls/ca/** 디렉터리에 CA를 복사합니다.

추가 리소스

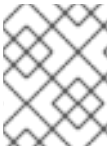
- Red Hat Ansible Automation Platform 시스템 요구 사항

3장. 자동화 메시 설계 패턴

이 섹션의 자동화 메시 토폴로지는 환경에서 메시 배포를 설계하는 데 사용할 수 있는 예를 제공합니다. 예를 들어 간단한 hydrid 노드 배포에서 다양한 자동화 컨트롤러 인스턴스를 배포하는 복잡한 패턴에 이르기까지 여러 실행 및 홉 노드를 사용하는 경우가 있습니다.

사전 요구 사항

- 노드 유형 및 관계에 대한 개념 정보를 검토했습니다.



참고

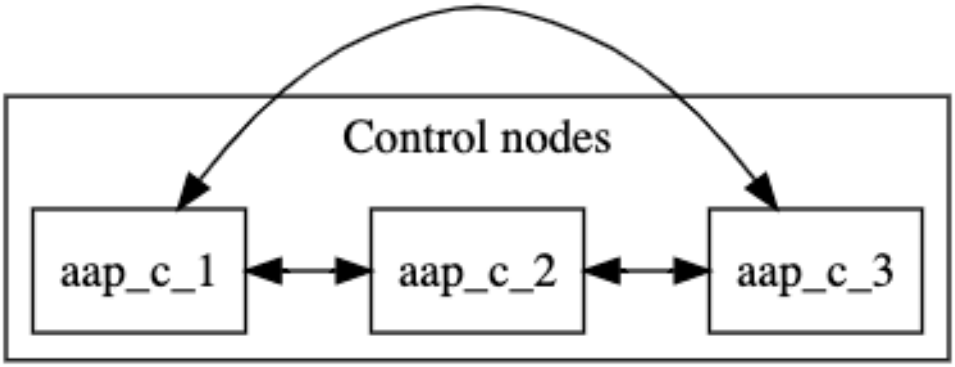
다음 예제에는 메시 토폴로지를 설명하는 이미지가 있습니다. 이미지의 화살표는 피어링의 방향을 나타냅니다. 피어링이 설정된 후 노드 간 연결은 양방향 통신을 허용합니다.

3.1. 여러 하이브리드 노드 인벤토리 파일 예

이 예제 인벤토리 파일은 여러 하이브리드 노드로 구성된 컨트롤 플레인을 배포합니다. 컨트롤 플레인의 노드는 자동으로 서로 피어링됩니다.

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

다음 이미지는 이 메시 네트워크의 토폴로지를 표시합니다.



컨트롤 플레인에서 노드의 기본 **node_type** 은 **하이브리드** 입니다. 개별 노드의 **node_type** 을 **[automationcontroller group]** 에서 **하이브리드** 로 명시적으로 설정할 수 있습니다.

```
[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
```

또는 **[automationcontroller]** 그룹에 있는 모든 노드의 **node_type** 을 설정할 수 있습니다. 컨트롤 플레인에 새 노드를 추가하면 자동으로 하이브리드 노드로 설정됩니다.

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

```
aap_c_3.example.com

[automationcontroller:vars]
node_type=hybrid
```

나중에 컨트롤 플레인에 컨트롤 노드를 추가할 수 있다고 생각되면 하이브리드 노드에 대한 별도의 그룹을 정의하고 그룹에 대한 **node_type** 을 설정하는 것이 좋습니다.

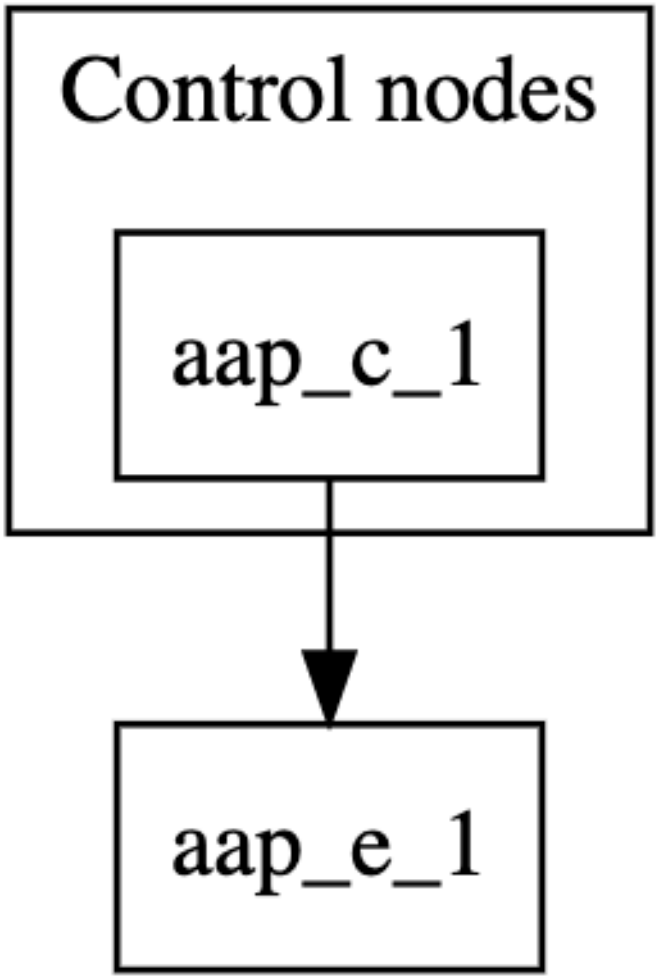
```
[automationcontroller]
aap_c_1.example.com
aap_c_2.example.com
aap_c_3.example.com

[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. 단일 실행 노드가 있는 단일 노드 컨트롤 플레인

이 예제 인벤토리 파일은 단일 노드 컨트롤 플레인을 배포하고 실행 노드와의 피어 관계를 설정합니다.

```
[automationcontroller]
aap_c_1.example.com

[automationcontroller:vars]
node_type=control
peers=execution_nodes

[execution_nodes]
aap_e_1.example.com
```

다음 이미지는 이 메시 네트워크의 토폴로지를 표시합니다.



[automationcontroller] 스탠자는 제어 노드를 정의합니다. 새 노드를 Automationcontroller 그룹에 추가하면 **aap_c_1.example.com** 노드와 자동으로 피어링됩니다.

[automationcontroller:vars] 스탠자는 **컨트롤** 플레인의 모든 노드에 대해 노드 유형을 제어하도록 설정하고 실행 노드와 실행 노드를 연결하는 방법을 정의합니다.

- 새 노드를 **execution_nodes** 그룹에 추가하면 컨트롤 플레인 노드가 자동으로 피어링됩니다.
- **자동화 컨트롤러** 그룹에 새 노드를 추가하면 노드 유형이 **control** 로 설정됩니다.

[execution_nodes] 스탠자는 인벤토리의 모든 실행 및 홉 노드를 나열합니다. 기본 노드 유형은 **execution** 입니다. 개별 노드의 노드 유형을 지정할 수 있습니다.

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

또는 **[execution_nodes]** 그룹에 있는 모든 실행 노드의 node_type을 설정할 수 있습니다. 그룹에 새 노드를 추가하면 자동으로 실행 노드로 설정됩니다.

```
[execution_nodes]
aap_e_1.example.com

[execution_nodes:vars]
node_type=execution
```

나중에 인벤토리에 홉 노드를 추가하려면 실행 노드에 대한 별도의 그룹을 정의하고 그룹에 대한 **node_type** 을 설정하는 것이 좋습니다.

```
[execution_nodes]
aap_e_1.example.com

[local_execution_group]
aap_e_1.example.com

[local_execution_group:vars]
node_type=execution
```

3.3. 최소 탄력적 구성

이 예제 인벤토리 파일은 두 개의 컨트롤 노드와 두 개의 실행 노드로 구성된 컨트롤 플레인을 배포합니다. 컨트롤 플레인의 모든 노드는 자동으로 서로 피어링됩니다. 컨트롤 플레인의 모든 노드는 **execution_nodes** 그룹의 모든 노드와 피어링됩니다. 이 구성은 모든 제어 노드에서 실행 노드에 연결할 수 있으므로 탄력적입니다.

용량 알고리즘은 작업이 시작될 때 선택한 제어 노드를 결정합니다. 자세한 내용은 자동화 컨트롤러 *사용자 가이드의 자동화 컨트롤러 용량 결정 및 작업에 미치는 영향을* 참조하십시오.

다음 인벤토리 파일은 이 구성을 정의합니다.

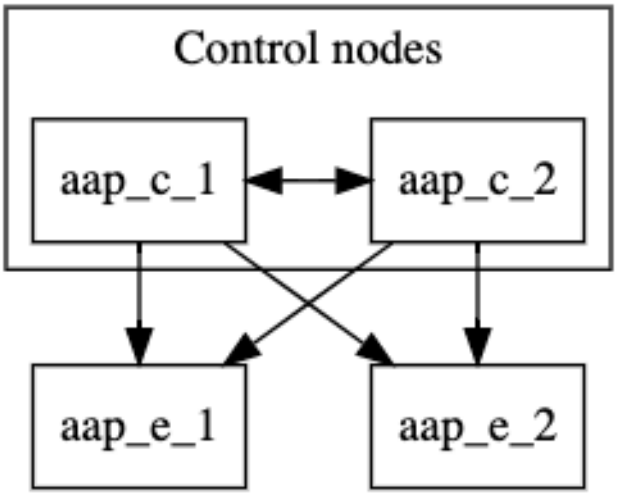
```
[automationcontroller]
aap_c_1.example.com
aap_c_2.example.com

[automationcontroller:vars]
node_type=control
peers=execution_nodes

[execution_nodes]
aap_e_1.example.com
aap_e_2.example.com
```

[automationcontroller] 스탠자는 제어 노드를 정의합니다. 컨트롤 플레인의 모든 노드는 서로 피어링됩니다. 새 노드를 Automation **controller** 그룹에 추가하면 원래 노드와 자동으로 피어링됩니다.

[automationcontroller:vars] 스탠자는 **컨트롤** 플레인의 모든 노드에 대해 노드 유형을 제어하도록 설정하고 실행 노드와 실행 노드를 연결하는 방법을 정의합니다.

- 새 노드를 **execution_nodes** 그룹에 추가하면 컨트롤 플레인 노드가 자동으로 피어링됩니다.
- **자동화 컨트롤러** 그룹에 새 노드를 추가하면 노드 유형이 **control** 로 설정됩니다.

다음 이미지는 이 메시 네트워크의 토폴로지를 표시합니다.



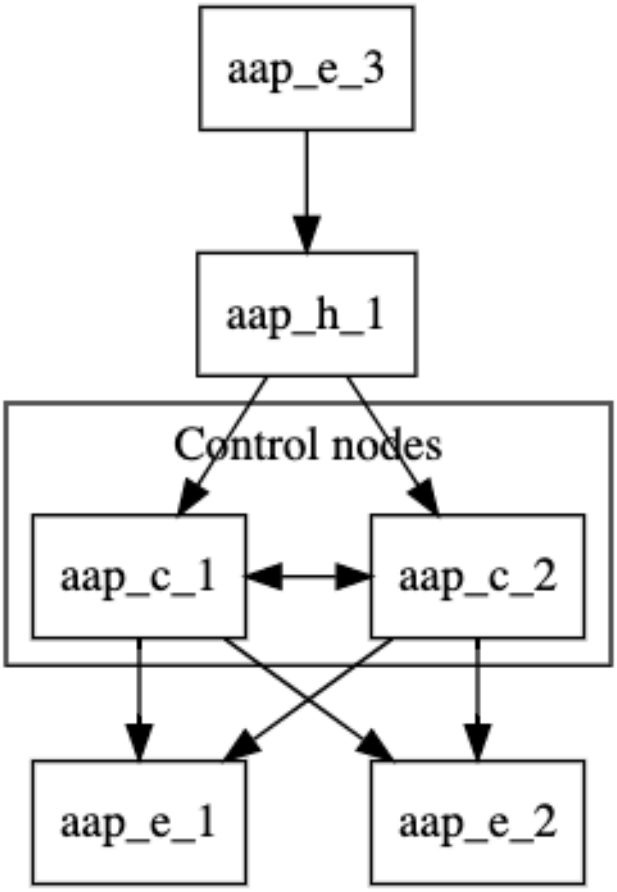
3.4. 분리된 로컬 및 원격 실행 구성

이 구성은 홉 노드와 원격 실행 노드를 탄력적 구성에 추가합니다. 홉 노드에서 원격 실행 노드에 연결할 수 있습니다.

원격 위치에서 실행 노드를 설정하거나 DMZ 네트워크에서 자동화를 실행해야 하는 경우 이 설정을 사용할 수 있습니다.

```
[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 node_type=hop
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
```

다음 이미지는 이 메시 네트워크의 토폴로지를 표시합니다.



[automationcontroller:vars] 스탠자는 컨트롤 플레인의 모든 노드의 노드 유형을 설정하고 컨트롤 노드가 로컬 실행 노드와 통신하는 방법을 정의합니다.

- 컨트롤 플레인의 모든 노드는 자동으로 서로 피어링됩니다.
- 컨트롤 플레인의 모든 노드는 모든 로컬 실행 노드와 피어링됩니다.

노드 그룹의 이름이 **instance_group_** 로 시작되면 설치 프로그램에서 인스턴스 그룹으로 인식하고 Ansible Automation Platform 사용자 인터페이스에 추가합니다.

3.5. MULTI-HOPPED 실행 노드

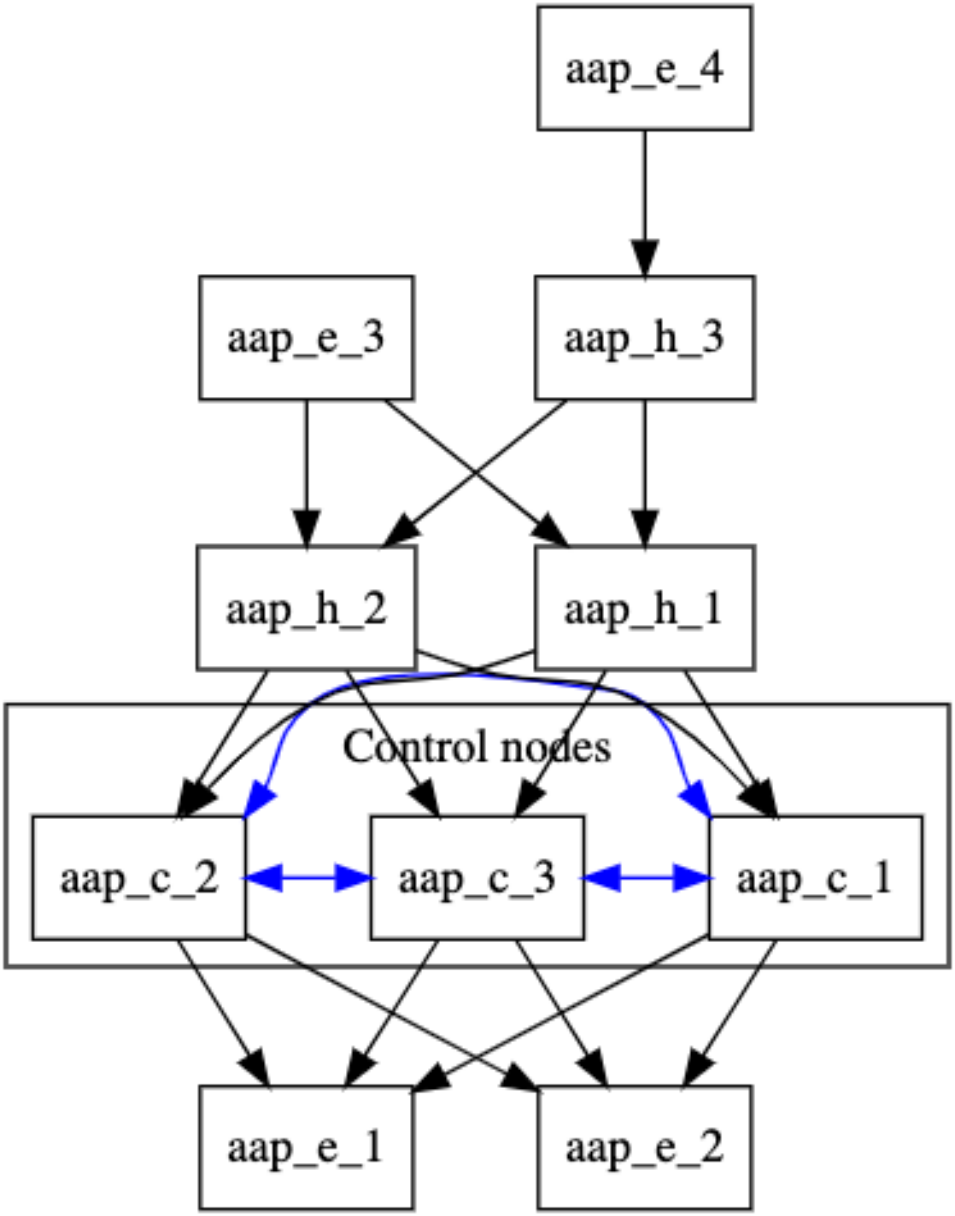
이 구성에서 탄력적 컨트롤러 노드는 탄력적인 로컬 실행 노드로 피어링됩니다. 탄력적 인 로컬 홉 노드는 컨트롤러 노드와 피어링됩니다. 원격 실행 노드와 원격 홉 노드는 로컬 홉 노드와 피어링됩니다.

원격 네트워크에서 DMZ 네트워크에서 자동화를 실행해야 하는 경우 이 설정을 사용할 수 있습니다.

```
[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
```

다음 이미지는 이 메시 네트워크의 토폴로지를 표시합니다.



[automationcontroller:vars] 스탠자는 컨트롤 플레인의 모든 노드의 노드 유형을 설정하고 컨트롤 노드가 로컬 실행 노드와 통신하는 방법을 정의합니다.

- 컨트롤 플레인의 모든 노드는 자동으로 서로 피어링됩니다.
- 컨트롤 플레인의 모든 노드는 모든 로컬 실행 노드와 피어링됩니다.

[local_hop:vars] 스탠자는 **[local_hop]** 그룹의 모든 노드를 모든 컨트롤 노드와 피어링합니다.

노드 그룹의 이름이 **instance_group_** 로 시작되면 설치 프로그램에서 인스턴스 그룹으로 인식하고 Ansible Automation Platform 사용자 인터페이스에 추가합니다.

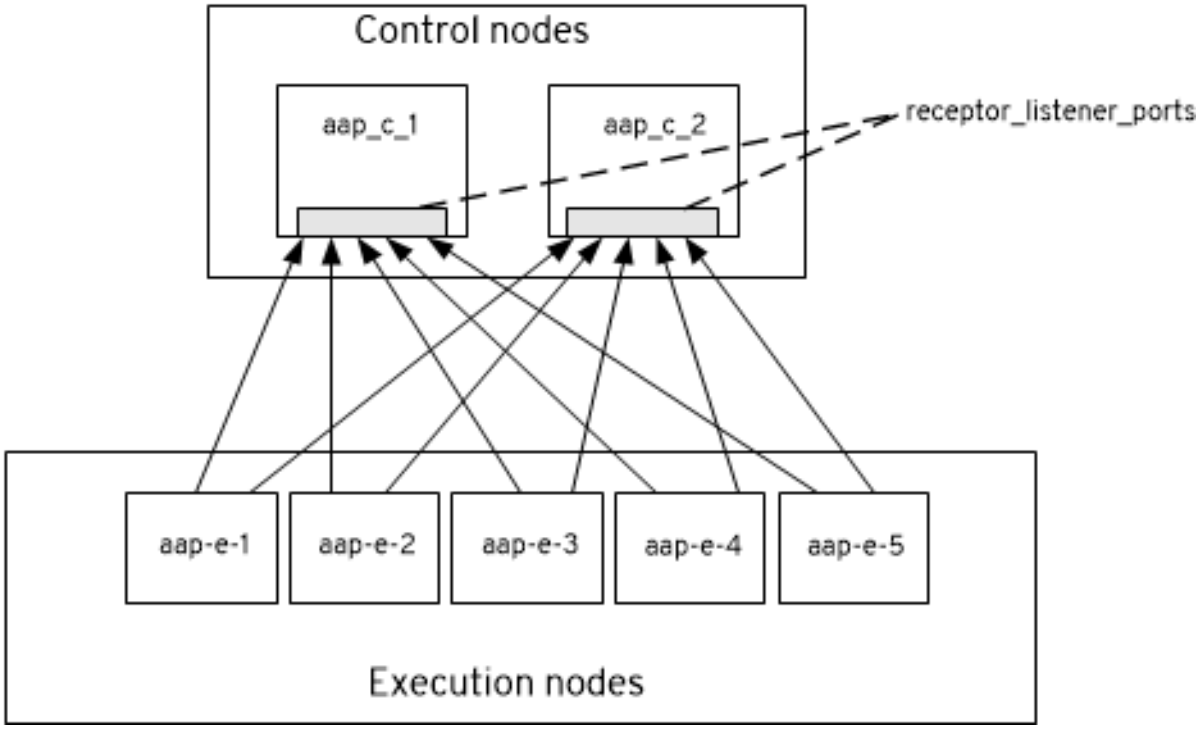
3.6. 컨트롤러 노드에 대한 아웃 바운드 연결만

이 예제 인벤토리 파일은 두 개의 컨트롤 노드와 여러 실행 노드로 구성된 컨트롤 플레인을 배포합니다. 컨트롤러 노드에 대한 아웃바운드 연결만 'execution_nodes' 그룹의 모든 노드는 컨트롤러 플레인의 모든 노드와 피어링됩니다.

```
[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
```

다음 이미지는 이 메시 네트워크의 토폴로지를 표시합니다.

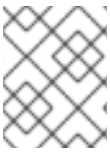


4장. 개별 노드 또는 그룹 프로비저닝 해제

Ansible Automation Platform 설치 프로그램을 사용하여 자동화 메시 노드 및 인스턴스 그룹을 프로비저닝 해제할 수 있습니다. 이 섹션의 절차에서는 각 프로시저에 대한 인벤토리 파일 예제를 사용하여 특정 노드 또는 전체 그룹을 프로비저닝하는 방법을 설명합니다.

4.1. 설치 프로그램을 사용하여 개별 노드 프로비저닝 해제

Ansible Automation Platform 설치 프로그램을 사용하여 자동화 메시에서 노드를 프로비저닝 해제할 수 있습니다. **인벤토리** 파일을 편집하여 노드를 프로비저닝 해제하도록 표시한 다음 설치 프로그램을 실행합니다. 설치 프로그램을 실행하면 노드에 연결된 모든 구성 파일과 로그도 제거됩니다.



참고

[automationcontroller] 그룹에 지정된 첫 번째 호스트를 제외하고 인벤토리의 호스트를 프로비저닝 해제할 수 있습니다.

프로세스

- 프로비저닝 해제하려는 설치 파일의 노드에 **node_state=deprovision** 을 추가합니다.

예제

이 예제 인벤토리 파일은 자동화 메시 구성에서 두 개의 노드를 프로비저닝 해제합니다.

```
[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. 격리된 노드 프로비저닝 해제

awx-manage deprovisioning 유틸리티를 사용하여 격리된 노드를 수동으로 제거하는 옵션이 있습니다.

주의

프로비저닝 해제 명령을 사용하여 실행 노드로 마이그레이션되지 않은 격리된 노드만 제거합니다. 자동화 메시 아키텍처에서 실행 노드를 프로비저닝 해제하려면 대신 설치 프로그램 방법을 사용합니다.

프로세스

1. 인스턴스를 종료합니다.

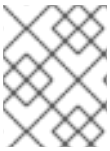
```
$ automation-controller-service stop
```

2. 다른 인스턴스에서 프로비저닝 해제 명령을 실행하여 ***host_name*** 을 인벤토리 파일에 나열된 대로 노드 이름으로 교체합니다.

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

4.2. 설치 프로그램을 사용하여 그룹 프로비저닝 해제

Ansible Automation Platform 설치 프로그램을 사용하여 자동화 메시에서 전체 그룹을 프로비저닝 해제할 수 있습니다. 설치 프로그램을 실행하면 모든 구성 파일과 그룹의 노드에 연결된 로그가 제거됩니다.



참고

[automationcontroller] 그룹에 지정된 첫 번째 호스트를 제외하고 인벤토리의 모든 호스트를 프로비저닝 해제할 수 있습니다.

프로세스

- 프로비저닝 해제하려는 그룹과 연결된 [group:vars]에 **node_state=deprovision** 을 추가합니다.

예제

```
[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. 격리된 인스턴스 그룹 프로비저닝 해제

awx-manage deprovisioning 유틸리티를 사용하여 격리된 인스턴스 그룹을 수동으로 제거하는 옵션이 있습니다.

주의

프로비저닝 해제 명령을 사용하여 격리된 인스턴스 그룹만 제거합니다. 자동화 메시 아키텍처에서 인스턴스 그룹을 프로비저닝 해제하려면 대신 설치 프로그램 방법을 사용합니다.

프로세스

- 다음 명령을 실행하여 < ***name&*** gt;을 인스턴스 그룹의 이름으로 바꿉니다.

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