Red Hat Ansible Automation Platform 2.1

Red Hat Ansible Automation Platform automation mesh guide

This guide provides information used to deploy automation mesh as part of your Ansible Automation Mesh Platform environment

Last Updated: 2022-08-23

Red Hat Ansible Automation Platform 2.1 Red Hat Ansible Automation Platform automation mesh guide

This guide provides information used to deploy automation mesh as part of your Ansible Automation Mesh Platform environment

Abstract

Providing Feedback: If you have a suggestion to improve this documentation, or find an error, please contact technical support at to create an issue on the Ansible Automation Platform Jira project using the Docs component.

Table of Contents

PREFACE 3

MAKING OPEN SOURCE MORE INCLUSIVE 4

CHAPTER 1. PLANNING FOR AUTOMATION MESH IN YOUR RED HAT ANSIBLE AUTOMATION PLATFORM ENVIRONMENT 5
1.1. ABOUT AUTOMATION MESH 5
1.2. CONTROL AND EXECUTION PLANES 5
1.2.1. Control plane 5
1.2.2. Execution plane 6
1.2.3. Peers 6
1.2.4. Defining automation mesh node types 6

CHAPTER 2. SETTING UP AUTOMATION MESH 8
2.1. AUTOMATION MESH INSTALLATION 8
2.2. IMPORTING A CERTIFICATE AUTHORITY (CA) CERTIFICATE 8

CHAPTER 3. AUTOMATION MESH DESIGN PATTERNS 9
3.1. SINGLE HYBRID NODE INVENTORY FILE EXAMPLE 9
3.2. MULTIPLE HYBRID NODES INVENTORY FILE EXAMPLE 9
3.3. SINGLE NODE CONTROL PLANE WITH SINGLE EXECUTION NODE 9
3.4. STANDARD CONTROL PLANE (3 NODE) AND (N) EXECUTION NODES, FULLY CONNECTED 9
3.5. STANDARD CONTROL PLANE WITH SINGLE EGRESS TO THE EXECUTION TOPOLOGY 10
3.6. STANDARD CONTROL PLANE AND EXECUTION TOPOLOGY WITH HOP NODES 10
3.7. COMPLEX AUTOMATION MESH DESIGN EXAMPLE 11

CHAPTER 4. DEPROVISIONING INDIVIDUAL NODES OR GROUPS 13
4.1. DEPROVISIONING INDIVIDUAL NODES USING THE INSTALLER 13
4.1.1. Deprovisioning isolated nodes 13
4.2. DEPROVISIONING GROUPS USING THE INSTALLER 14
4.2.1. Deprovisioning isolated instance groups 14

PREFACE

Thank you for your interest in Red Hat Ansible Automation Platform. Ansible Automation Platform is a commercial offering that helps teams manage complex multi-tier deployments by adding control, knowledge, and delegation to Ansible-powered environments.

This guide helps you to understand the installation requirements and processes behind installing Ansible Automation Platform. This document has been updated to include information for the latest release of Ansible Automation Platform.

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message .

CHAPTER 1. PLANNING FOR AUTOMATION MESH IN YOUR RED HAT ANSIBLE AUTOMATION PLATFORM ENVIRONMENT

The following topics contain information to help plan an automation mesh deployment in your Ansible Automation Platform environment. The subsequent sections explain the concepts that comprise automation mesh in addition to providing examples on how you can design automation mesh topologies. Simple to complex topology examples are included to illustrate the various ways you can deploy automation mesh.

1.1. ABOUT AUTOMATION MESH

Automation mesh is an overlay network intended to ease the distribution of work across a large and dispersed collection of workers through nodes that establish peer-to-peer connections with each other using existing networks.

Red Hat Ansible Automation Platform 2 replaces Ansible Tower and isolated nodes with automation controller and automation hub. Automation controller provides the control plane for automation through its UI, Restful API, RBAC, workflows and CI/CD integration, while Automation Mesh can be used for setting up, discovering, changing or modifying the nodes that form the control and execution layers.

Automation Mesh introduces:

- Dynamic cluster capacity that scales independently, allowing you to create, register, group, ungroup and deregister nodes with minimal downtime.
- Control and execution plane separation that enables you to scale playbook execution capacity independently from control plane capacity.
- Deployment choices that are resilient to latency, reconfigurable without outage, and that dynamically re-reroute to choose a different path when outages may exist. mesh routing changes.
- Connectivity that includes bi-directional, multi-hopped mesh communication possibilities which are Federal Information Processing Standards (FIPS) compliant.

1.2. CONTROL AND EXECUTION PLANES

Automation mesh makes use of unique node types to create both the **control** and **execution** plane. Learn more about the control and execution plane and their node types before designing your automation mesh topology.

1.2.1. Control plane

The **control plane** consists of hybrid and control nodes. Instances in the control plane run persistent automation controller services such as the the web server and task dispatcher, in addition to project updates, and management jobs.

- **Hybrid nodes** - this is the default node type for control plane nodes, responsible for automation controller runtime functions like project updates, management jobs and **ansible-runner** task operations. Hybrid nodes are also used for automation execution.
- **Control nodes** - control nodes run project and inventory updates and system jobs, but not regular jobs. Execution capabilities are disabled on these nodes.

1.2.2. Execution plane

The **execution plane** consists of execution nodes that execute automation on behalf of the control plane and have no control functions. Hop nodes serve to communicate. Nodes in the **execution plane** only run user-space jobs, and may be geographically separated, with high latency, from the control plane.

- **Execution nodes** - Execution nodes run jobs under **ansible-runner** with **podman** isolation. This node type is similar to isolated nodes.
- **Hop nodes** - similar to a jump host, hop nodes will route traffic to other execution nodes. Hop nodes cannot execute automation.

1.2.3. Peers

Peer relationships define node-to-node connections. You can define peers within the **[automationcontroller]** and **[execution_nodes]** groups or using the **[automationcontroller:vars]** or **[execution_nodes:vars]** groups

1.2.4. Defining automation mesh node types

You can a define node type either by its default value assigned by the inventory group or by using the **node_type** host variable. Specify the **node_type** either as part of the inventory group or within the inventory **vars** group. This section provides examples of how you can define node types in your inventory file. Nodes in **[execution_nodes]** default execution node_type. Hybrid node types can be overridden to be control type via **node_type=control**. Execution node type can be overridden to be hope node type via **node_type=hop**

Hybrid node

Nodes in **[automationcontroller]** default to the **hybrid node** type. In the below example, we create a single hybrid node:

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

Control node

Convert hybrid node types to control nodes using **node_type=control**:

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

Execution node

Nodes in the`[execution_nodes]` inventory group default to the **execution node** type. In the below example, we create a single execution node:

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

Hop node

Convert execution nodes to hop nodes using **node_type=control**:

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

Peer connections

Create node-to-node connections using the **peers=** host variable. The following example connects **control-plane-1.example.com** to **execution-node-1.example.com** and **execution-node-1.example.com** to **execution-node-2.example.com**:

```
[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
```

Additional resources

- See the example automation mesh topologies in this guide for more examples of how to implement mesh nodes.

CHAPTER 2. SETTING UP AUTOMATION MESH

Configure the Ansible Automation Platform installer to set up automation mesh for your Ansible environment. Perform additional tasks to customize your installation, such as importing a Certificate Authority (CA) certificate.

2.1. AUTOMATION MESH INSTALLATION

You use the Ansible Automation Platform installation program to set up automation mesh or to upgrade to automation mesh. To provide Ansible Automation Platform with details about the nodes, groups, and peer relationships in your mesh network, you define them in an the **inventory** file in the installer bundle.

Additional Resources

- Red Hat Ansible Automation Platform Installation Guide
- Automation Mesh Design Patterns

2.2. IMPORTING A CERTIFICATE AUTHORITY (CA) CERTIFICATE

A Certificate Authority (CA) verifies and signs individual node certificates in an automation mesh environment. You can provide your own CA by specifying the path to the certificate and the private RSA key file in the **inventory** file of your Red Hat Ansible Automation Platform installer.

> **NOTE**
>
> The Ansible Automation Platform installation program generates a CA if you do not provide one.

Procedure

1. Open the **inventory** file for editing.
2. Add the **mesh_ca_keyfile** variable and specify the full path to the private RSA key (**.key**).
3. Add the **mesh_ca_certfile** variable and specify the full path to the CA certificate file (**.crt**).
4. Save the changes to the inventory file.

Example

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

With the CA files added to the inventory file, run the installation program to apply the CA. This process copies the CA to the to **/etc/receptor/tls/ca/** directory on each control and execution node on your mesh network.

Additional resources

- Red Hat Ansible Automation Platform System Requirements

CHAPTER 3. AUTOMATION MESH DESIGN PATTERNS

The automation mesh topologies in this section provide examples you can use to design a mesh deployment in your environment. Examples range from a single, hydrid node deployment to a complex pattern that deploys numerous automation controller instances, employing several execution and hop nodes.

Prerequisites

- You reviewed conceptual information on node types and relationsips

3.1. SINGLE HYBRID NODE INVENTORY FILE EXAMPLE

This example inventory file deploys a single hybrid node control plane.

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

3.2. MULTIPLE HYBRID NODES INVENTORY FILE EXAMPLE

This example inventory file deploys a control plane consisting of multiple hybrid nodes.

```
[automationcontroller]
control-plane-1.example.com ansible_connection=local
control-plane-2.example.com
control-plane-3.example.com
```

3.3. SINGLE NODE CONTROL PLANE WITH SINGLE EXECUTION NODE

This example inventory file deploys a single node control plane and establishes a peer relationship to the execution node.

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

[automationcontroller:vars]
node_type=control
peers=execution_nodes

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

3.4. STANDARD CONTROL PLANE (3 NODE) AND (N) EXECUTION NODES, FULLY CONNECTED

This example inventory file deploys a control plane consisting of three nodes with fully connected peer execution nodes.

```
[automationcontroller]
control-plane-1.example.com
control-plane-2.example.com
```

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

[automationcontroller:vars]
node_type=control
peers=execution_nodes


[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_type=execution
```

3.5. STANDARD CONTROL PLANE WITH SINGLE EGRESS TO THE EXECUTION TOPOLOGY

This example inventory file deploys a control plane consisting of three nodes with a single point of egress to the execution plane.

```
[automationcontroller]
control-plane-1.example.com
control-plane-2.example.com
control-plane-3.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 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
```

3.6. STANDARD CONTROL PLANE AND EXECUTION TOPOLOGY WITH HOP NODES

This example inventory file deploys a control plane consisting of three nodes and an execution topology that utilizes hope nodes.

```
[automationcontroller]
control-plane-1.example.com
control-plane-2.example.com
control-plane-3.example.com
```

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

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

3.7. COMPLEX AUTOMATION MESH DESIGN EXAMPLE

This example inventory demonstrates a complex automation mesh design pattern that uses groups to associate nodes of the same type in order to reduce noise in the installer configuration. It deploys three automation controllers, two execution nodes and multiple hops with execution nodes.

NOTE

- The control plane contains only **control** nodes, which are auto-peered together.
- **[automationcontroller:vars]** peers the controllers to the *disconnected* execution nodes group **[instance_group_disconnected]**, ignoring the hop nodes in **[execution_nodes]**.
- All execution node types are defined in **[execution nodes]**, as well as which execution nodes need a hop. Hop nodes are also defined in this group.
- The **instance_grouo_prefix** automatically adds nodes into the automation controller user interface as an instance group after running the Red Hat Ansible Automation Platform installer.
- The **[local_hop]** group peers to the **[automationcontoller]** group.
- The **[remote_hop]** group peers to the **[local_hop]** group.

Inventory file

```
[automationcontroller]
control-plane-1.example.com ansible_connection=local
control-plane-2.example.com
control-plane-3.example.com

[automationcontroller:vars]
peers=instance_group_directconnected
node_type=control

[execution_nodes]
execution-node-1.example.com
execution-node-2.example.com
execution-node-3.example.com peers=local_hop
execution-node-4.example.com peers=remote_hop
hop-node-1.example.com node_type=hop
```

```
hop-node-2.example.com node_type=hop
hop-node-3.example.com node_type=hop

[instance_group_directconnected]
execution-node-1.example.com
execution-node-2.example.com

[instance_group_localhop]
execution-node-3.example.com

[instance_group_multihop]
execution-node-4.example.com

[local_hop]
hop-node-1.example.com
hop-node-2.example.com

[local_hop:vars]
peers=automationcontroller

[remote_hop]
hop-node-3.example.com

[remote_hop:vars]
peers=local_hop
```

CHAPTER 4. DEPROVISIONING INDIVIDUAL NODES OR GROUPS

You can deprovision automation mesh nodes and instance groups using the Ansible Automation Platform installer. The procedures in this section describe how to deprovision specific nodes or entire groups, with example inventory files for each procedure.

4.1. DEPROVISIONING INDIVIDUAL NODES USING THE INSTALLER

You can deprovision nodes from your automation mesh using the Ansible Automation Platform installer. Edit the **inventory** file to mark the nodes to deprovision, then run the installer. Running the installer also removes all configuration files and logs attached to the node.

NOTE

You can deprovision any of your inventory's hosts except for the first host specified in the **[automationcontroller]** group.

Procedure

- Append **node_state=deprovision** to nodes in the installer file you want to deprovision.

Example

This example inventory file deprovisions two nodes from an automation mesh configuration.

```
[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. Deprovisioning isolated nodes

You have the option to manually remove any isolated nodes using the **awx-manage** deprovisioning utility.

> **WARNING**
>
> Use the deprovisioning command to remove only isolated nodes that have not migrated to execution nodes. To deprovision execution nodes from your automation mesh architecture, use the installer method instead.

Procedure

1. Shut down the instance:

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

2. Run the deprovision command from another instance, replacing ***host_name*** with the name of the node as listed in the inventory file:

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

4.2. DEPROVISIONING GROUPS USING THE INSTALLER

You can deprovision entire groups from your automation mesh using the Ansible Automation Platform installer. Running the installer will remove all configuration files and logs attached to the nodes in the group.

> **NOTE**
>
> You can deprovision any hosts in your inventory except for the first host specified in the **[automationcontroller]** group.

Procedure

- Add **node_state=deprovision** to the [group:vars] associated with the group you wish to deprovision.

Example

```
[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. Deprovisioning isolated instance groups

You have the option to manually remove any isolated instance groups using the **awx-manage** deprovisioning utility.

> **WARNING**
>
> Use the deprovisioning command to only remove isolated instance groups. To deprovision instance groups from your automation mesh architecture, use the installer method instead.

Procedure

- Run the following command, replacing ***<name>*** with the name of the instance group:

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