(/rol/app/courses/do467-2.2/pages/ch07s08)
(/rol/app/courses/do467-2.2/pages/ch08s02)
(/rol/app/courses/do467-2.2/pages/ch08s04)
(/rol/app/courses/do467-2.2/pages/ch08s06)
(/rol/app/courses/do467-2.2/pages/ch08s08)
(/rol/app/courses/do467-2.2/pages/ch09s02)
(/rol/app/courses/do467-2.2/pages/ch09s04)
(/rol/app/courses/do467-2.2/pages/ch09s06)
(/rol/app/courses/do467-2.2/pages/ch10s02)
(/rol/app/courses/do467-2.2/pages/ch10s04)
(/rol/app/courses/do467-2.2/pages/ch10s06)
(/rol/app/courses/do467-2.2/pages/ch10s08)
(/rol/app/courses/do467-2.2/pages/ch11)
(/rol/app/courses/do467-2.2/pages/ch11s03)
(/rol/app/courses/do467-2.2/pages/ch11s05)
(/rol/app/courses/do467-2.2/pages/ch11s07)
(/rol/app/courses/do467-2.2/pages/ch12)
(/rol/app/courses/do467-2.2/pages/ch12s03)
(/rol/app/courses/do467-2.2/pages/ch12s05)
(/rol/app/courses/do467-2.2/pages/ch12s07)
8 (/rol/app/courses/do467-2.2/pages/ch08)
(/rol/app/courses/do467-2.2/pages/ch08s03)
(/rol/app/courses/do467-2.2/pages/ch08s05)
(/rol/app/courses/do467-2.2/pages/ch08s07)
9 (/rol/app/courses/do467-2.2/pages/ch09)
(/rol/app/courses/do467-2.2/pages/ch09s03)
(/rol/app/courses/do467-2.2/pages/ch09s05)
10 (/rol/app/courses/do467-2.2/pages/ch10)
(/rol/app/courses/do467-2.2/pages/ch10s03)
(/rol/app/courses/do467-2.2/pages/ch10s05)
(/rol/app/courses/do467-2.2/pages/ch10s07)
(/rol/app/courses/do467-2.2/pages/ch10s09)
(/rol/app/courses/do467-2.2/pages/ch11s02)
(/rol/app/courses/do467-2.2/pages/ch11s04)
(/rol/app/courses/do467-2.2/pages/ch11s06)
(/rol/app/courses/do467-2.2/pages/ch11s08)
(/rol/app/courses/do467-2.2/pages/ch12s02)
(/rol/app/courses/do467-2.2/pages/ch12s04)
(/rol/app/courses/do467-2.2/pages/ch12s06)
(/rol/app/courses/do467-2.2/pages/ch12s08)
(/rol/app/courses/do467-2.2/pages/ch12s09)
11 (/ro /a 2.2/pag
12 (/ro /a 2.2/pag
(/rol/app/courses/do467-2.2/pages/ch11s02) < Previous
(/rol/app/cour Next >

Deploying Distributed Execution with Automation Mesh

Objectives

- Distribute execution of Ansible Playbooks from automation controller control or hybrid nodes to remote execution nodes, communicating with them using automation mesh.

Configuring Automation Mesh

You can configure automation mesh to separate execution nodes from your control nodes, so that you can provide resilience and scalability to your automation, and so that you can move execution of your automation closer to your managed hosts.

To configure Red Hat Ansible Automation Platform nodes that are connected by automation mesh, edit the inventory file that is used by the installation script.

The following example deploys one control node (`controller.lab.example.com`) and one execution node (`exec1.lab.example.com`).

```
[automationcontroller] (1)
controller.lab.example.com

[execution_nodes] (2)
exec1.lab.example.com

[automationcontroller:vars] (3)
peers=execution_nodes (4)
node_type=control (5)
```

1. To add hybrid or control nodes, add an entry for each node in the `automationcontroller` group. By default, hosts in this group are hybrid nodes (`node_type=hybrid`).
2. To add execution nodes or hop nodes, add an entry for each node in the `execution_nodes` group. By default, hosts in this group are execution nodes (`node_type=execution`).
3. Use group variables to set up definitions, such as `peers` and `node_type`, for all nodes in the group. You can also set the definitions individually on each host entry, but if you are repeating the configuration, then it is easier to set up a group and define variables.
4. The `peers=execution_nodes` directive specifies that all nodes in the `automationcontroller` group have a direct peering relationship in automation mesh with all nodes in `execution_nodes` (only `exec1.lab.example.com` in this example).
5. The `node_type=control` directive specifies that all nodes in the `automationcontroller` group be control nodes instead of hybrid nodes. You can override settings applied through group variables by setting variables for each individual host.

Creating Instance Groups

By default, automation mesh creates the `controlplane` instance group for nodes in the `[automationcontroller]` section of the inventory file, and the `default` instance group for `execution` nodes in the `[execution_nodes]` section of the inventory file.

You can configure the inventory to create additional instance groups. For example, you might want to create an instance group of `execution` nodes that are local to a particular data center. If you create instance groups based on data centers, then you can configure an inventory or a job template to run jobs for managed hosts that are in the same data center as the `execution` nodes.

The installer automatically creates instance groups for any group names in the installer's inventory with the `instance_group_` prefix. All of the hosts in the instance group must be execution nodes and cannot be hop nodes.

The following fragment of an inventory file creates two instance groups when you run `setup.sh`: `local`, which consists of the execution nodes `exec1` and `exec2`, and `remote`, which consists of the execution node `exec3`.

```
[instance_group_local]
exec1.lab.example.com
exec2.lab.example.com

[instance_group_remote]
exec3.lab.example.com
```

You can also manage instance groups and assign execution nodes to them in the automation controller web UI.

Adding Nodes to Automation Mesh

To add an additional node, add the new node to the inventory file and run the installation script again.

Removing Nodes from Automation Mesh

To remove a node, append `node_state=deprovision` to the node entry in the inventory file and run the installation script again.

```
[automationcontroller]
controller1.lab.example.com node_type=control
controller2.lab.example.com
controller3.lab.example.com node_state=deprovision

[execution_nodes]
exec1.lab.example.com
exec2.lab.example.com node_state=deprovision
```

IMPORTANT

You cannot use the `node_state=deprovision` variable with the first entry in the `[automationcontroller]` section because we need at least one controller for an operational Ansible Automation Platform. The installer uses the first entry to launch the remaining installation and configuration. If you want to use the `node_state=deprovision` variable with the host listed in the first entry, then move that line to a different position in the `[automationcontroller]` section.

Removing Groups from Automation Mesh

You can also remove entire groups from your automation mesh. The installer removes all configuration files and logs attached to the nodes in the group.

```
[execution_nodes]
exec1.lab.example.com peers=exec2.lab.example.com
exec2.lab.example.com peers=exec3.lab.example.com
exec3.lab.example.com

[execution_nodes:vars]
node_state=deprovision
```

Visualizing Automation Mesh Topology

The Red Hat Ansible Automation Platform 2 installer provides a way to visualize the automation mesh topology defined in your installation inventory file. Using your inventory file, the installer can generate a text file that shows the relationships between the different nodes in your automation mesh. You can then use the `dot` command, provided by the `graphviz` package, to render the automation mesh topology text file as a graphic file.

> **NOTE**
>
> Red Hat Ansible Automation Platform 2.2 adds a topology viewer in the automation controller web UI under **Administration → Topology View**.

The following steps create and visualize the mesh topology:

1. Use an existing inventory file or create a new one. This example uses an inventory with two hybrid control nodes and one execution node.

```
[automationcontroller]
controller1.lab.example.com
control2.lab.example.com

[automationcontroller:vars]
peers=execution_nodes

[execution_nodes]
exec1.lab.example.com
```

2. Execute the script to generate the `mesh-topology.dot` file.

```
[user@demo aap-bundle]$ ./setup.sh -- --tag generate_dot_file
```

> **NOTE**
>
> You can run the `./setup.sh --help` command to display command usage, including how to pass Ansible options.

3. Make sure that the `graphviz` RPM package is installed.

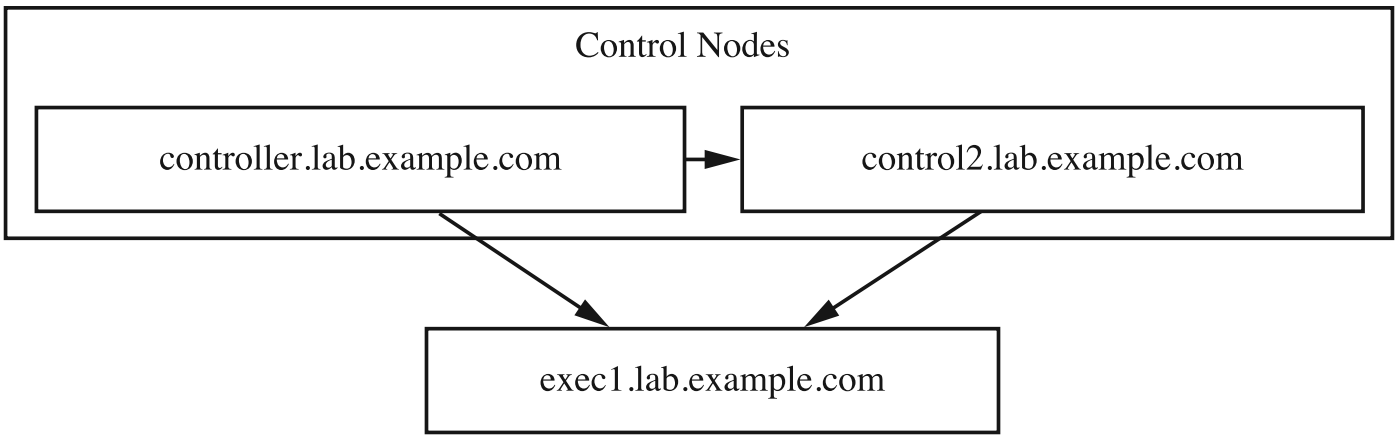
4. Render the generated topology file, `mesh-topology.dot`, as a graphic.

```
[user@demo aap-bundle]$ dot -Tjpg mesh-topology.dot \
> -o graph-topology.jpg
...output omitted...
```

NOTE

You can render the file in other formats by using the `-T` option to specify the output format, such as GIF, PNG, or SVG (`-Tgif`, `-Tpng`, or `-Tsvg`).

5. Open the generated `graph-topology.jpg` file with a web browser or other image viewer.



IMPORTANT

Even though the preceding diagram shows arrows for the connections between the various nodes on the automation mesh, these are peer connections and the arrow heads can be misleading.

Automation Mesh Design Patterns

Automation mesh is flexible and you can adapt it to meet your needs. The following two examples provide starting points. You might expand from these examples as you add locations in different regions, data centers, or behind firewalls. The key to setting up an effective automation mesh is to know the function of your nodes and which ones can directly peer with each other over your network.

Example 1: A Minimal Resilient Configuration

The minimal resilient automation mesh configuration provides resilience in the control and execution planes. The control nodes connect to each other and to each execution node. There is no single point of failure.

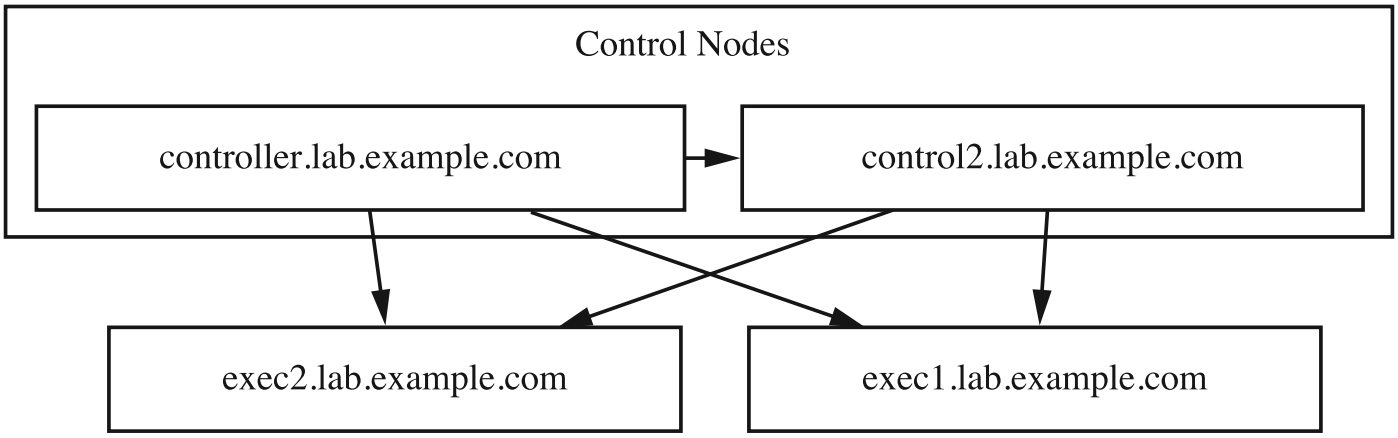


Figure 11.11: Minimal resilient configuration

The inventory file contains the following content:

```
[automationcontroller] (1)
controller.lab.example.com
control2.lab.example.com

[automationcontroller:vars]
node_type=control (2)
peers=execution_nodes (3)

[execution_nodes] (4)
exec1.lab.example.com
exec2.lab.example.com
```

(1) Two controllers manage the control plane.

(2) The controllers are `control` nodes instead of the default `hybrid` nodes.

(3) The controllers peer with each node in the `execution_nodes` group.

(4) Two execution nodes manage the execution plane.

Example 2: A Resilient Configuration with Local and Remote Instance Groups

This example modifies the minimal resilient configuration to add an additional `execution` node that is reached through a `hop` node. It also sets up two instance groups, `local` (consisting of `exec1` and `exec2`) and `remote` (consisting of `exec3`, which is behind the hop node). The instance groups are not shown on the following diagram.

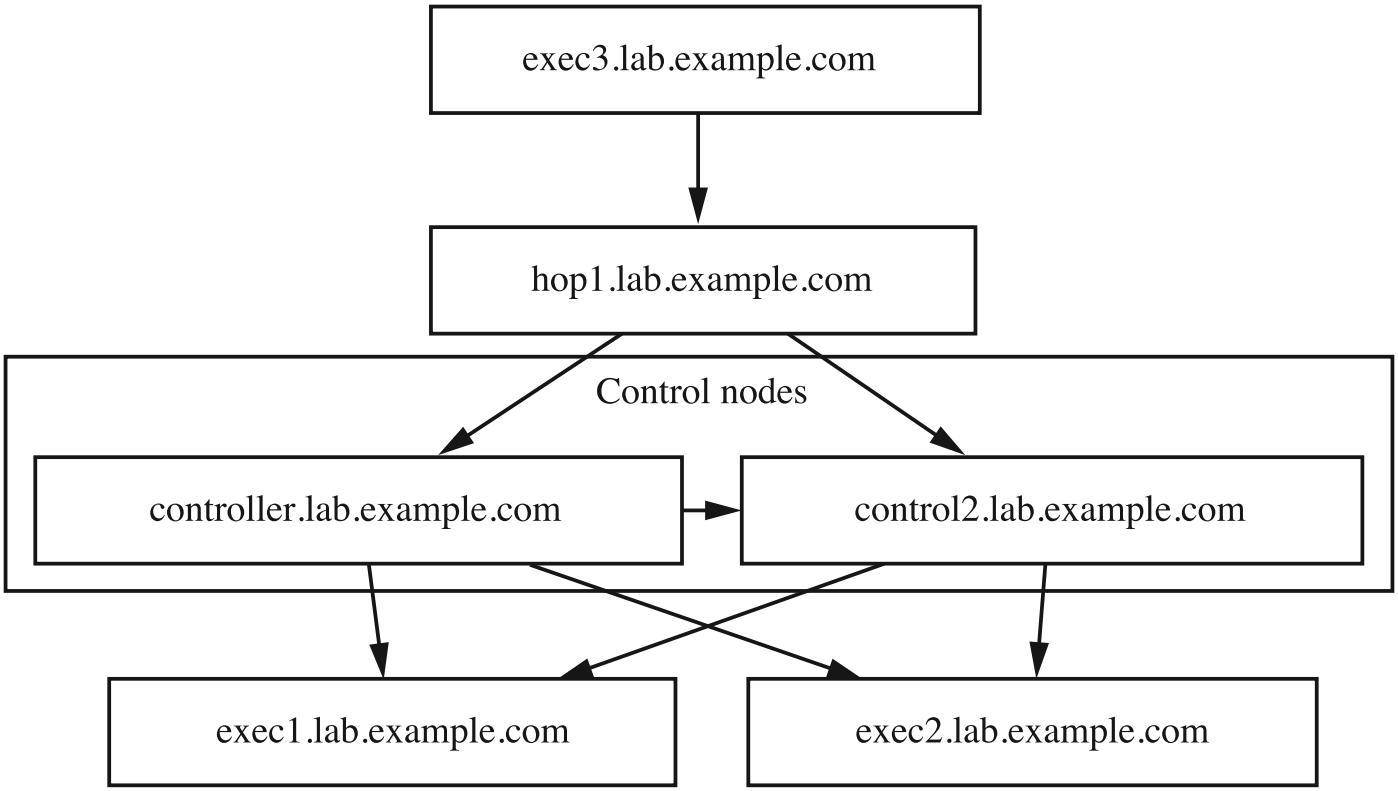


Figure 11.12: Local and remote execution

This excerpt is from the inventory file that sets up this configuration:

```
[automationcontroller]
controller.lab.example.com
control2.lab.example.com

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

[execution_nodes] (2)
exec1.lab.example.com
exec2.lab.example.com
exec3.lab.example.com
hop1.lab.example.com

[instance_group_local] (3)
exec1.lab.example.com
exec2.lab.example.com

[instance_group_remote] (4)
exec3.lab.example.com

[instance_group_remote:vars]
peers=hop (5)

[hop] (6)
hop1.lab.example.com

[hop:vars] (7)
node_type=hop
peers=automationcontroller
```

1. All of the controllers are `control` nodes. Because at least one execution node is a hop node, the `automationcontrollers` group cannot peer with the entire `execution_nodes` group.
2. List all execution nodes regardless of the node type. You could define the node type for each node, but it might be easier to create groups and group variables as shown in this example.
3. The four hosts defined in the `execution_nodes` group, only `exec1.lab.example.com` and `exec2.lab.example.com` can establish peer connections with hosts in the `automationcontrollers` group. Because of the `instance_group_` prefix, the installation script creates this instance group resource.
4. Execution nodes in the `instance_group_remote` group cannot directly connect to the control nodes.
5. The execution nodes in the `instance_group_remote` group peer with the hosts in the `hop` group.
6. Creating a separate group for `hop` nodes makes it easier to assign common variables and add or remove hop nodes from the group.
7. All of the hosts in the `hop` group are hop nodes and they peer with all hosts in the `automationcontroller` group.

Validation Checks

Prior to installation, the installer script runs validation checks on your defined automation mesh configuration.

- A host cannot belong to both the `[automationcontroller]` and `[execution_nodes]` groups.
- A host cannot peer to a node that does not exist.
- A host cannot peer to itself.
- A host cannot have an inbound and an outbound connection to the same nodes.
- Execution nodes must have a path back to the control plane.

REFERENCES

Red Hat Ansible Automation Platform Automation Mesh Guide (https://access.redhat.com/documentation/en-us/red_hat_ansible_automation_platform/2.1/html-single/red_hat_ansible_automation_platform_automation_mesh_guide/index)

(/rol/app/courses/do467-2.2/pages/ch11s02) Previous

(/rol/app/cour Next

Revision: do467-2.2-08877c1