Red Hat Enterprise Linux 8

Administration and configuration tasks using System Roles in RHEL

Applying RHEL System Roles using Red Hat Ansible Automation Platform playbooks to perform system administration tasks

Last Updated: 2022-11-16

Red Hat Enterprise Linux 8 Administration and configuration tasks using System Roles in RHEL

Applying RHEL System Roles using Red Hat Ansible Automation Platform playbooks to perform system administration tasks

Legal Notice

Copyright © 2022 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.

Abstract

This document describes configuring system roles using Ansible on Red Hat Enterprise Linux 8. The title focuses on: the RHEL System Roles are a collection of Ansible roles, modules, and playbooks that provide a stable and consistent configuration interface to manage and configure Red Hat Enterprise Linux. They are designed to be forward compatible with multiple major release versions of Red Hat Enterprise Linux 8.

Table of Contents

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.

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our documentation. Let us know how we can improve it.

Submitting comments on specific passages

1. View the documentation in the **Multi-page HTML** format and ensure that you see the **Feedback** button in the upper right corner after the page fully loads.
2. Use your cursor to highlight the part of the text that you want to comment on.
3. Click the **Add Feedback** button that appears near the highlighted text.
4. Add your feedback and click **Submit**.

Submitting feedback through Bugzilla (account required)

1. Log in to the Bugzilla website.
2. Select the correct version from the **Version** menu.
3. Enter a descriptive title in the **Summary** field.
4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
5. Click **Submit Bug**.

CHAPTER 1. PREPARING A CONTROL NODE AND MANAGED NODES TO USE RHEL SYSTEM ROLES

Before you can use individual RHEL System Roles to manage services and settings, prepare the involved hosts.

1.1. INTRODUCTION TO RHEL SYSTEM ROLES

RHEL System Roles is a collection of Ansible roles and modules. RHEL System Roles provide a configuration interface to remotely manage multiple RHEL systems. The interface enables managing system configurations across multiple versions of RHEL, as well as adopting new major releases.

On Red Hat Enterprise Linux 8, the interface currently consists of the following roles:

- Certificate Issuance and Renewal (**certificate**)
- Cockpit (**cockpit**)
- Firewalld (**firewall**)
- HA Cluster (**ha_cluster**)
- Kernel Dumps (**kdump**)
- Kernel Settings (**kernel_settings**)
- Logging (**logging**)
- Metrics (PCP) (**metrics**)
- Microsoft SQL Server (**microsoft.sql.server**)
- Networking (**network**)
- Network Bound Disk Encryption client and Network Bound Disk Encryption server (**nbde_client** and **nbde_server**)
- Postfix (**postfix**)
- SELinux (**selinux**)
- SSH client (**ssh**)
- SSH server (**sshd**)
- Storage (**storage**)
- Terminal Session Recording (**tlog**)
- Time Synchronization (**timesync**)
- VPN (**vpn**)

All these roles are provided by the **rhel-system-roles** package available in the **AppStream** repository.

Additional resources

- Red Hat Enterprise Linux (RHEL) System Roles
- **/usr/share/doc/rhel-system-roles/** provided by the **rhel-system-roles** package.

1.2. RHEL SYSTEM ROLES TERMINOLOGY

You can find the following terms across this documentation:

Ansible playbook

Playbooks are Ansible's configuration, deployment, and orchestration language. They can describe a policy you want your remote systems to enforce, or a set of steps in a general IT process.

Control node

Any machine with Ansible installed. You can run commands and playbooks, invoking /usr/bin/ansible or /usr/bin/ansible-playbook, from any control node. You can use any computer that has Python installed on it as a control node - laptops, shared desktops, and servers can all run Ansible. However, you cannot use a Windows machine as a control node. You can have multiple control nodes.

Inventory

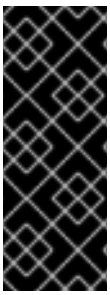
A list of managed nodes. An inventory file is also sometimes called a "hostfile". Your inventory can specify information like IP address for each managed node. An inventory can also organize managed nodes, creating and nesting groups for easier scaling. To learn more about inventory, see the Working with Inventory section.

Managed nodes

The network devices, servers, or both that you manage with Ansible. Managed nodes are also sometimes called "hosts". Ansible is not installed on managed nodes.

1.3. PREPARING A CONTROL NODE

RHEL includes **Ansible Core** in the **AppStream** repository with a limited scope of support. If you require additional support for Ansible, contact Red Hat to learn more about the **Ansible Automation Platform** subscription.



IMPORTANT

On RHEL 8.6 and later, use **Ansible Core** if you do not have a **Red Hat Ansible Automation Platform** subscription. Do not install **Ansible Engine** from the **ansible-2-for-rhel-8-x86_64-rpms** repository. Packages in this repository might not be compatible with Ansible automation content in RHEL 8.6 and later. Additionally, Red Hat does not provide security updates and bug fixes for **Ansible Engine** for the same amount of time as for RHEL. For further details, see Using Ansible in RHEL 8.6 and later .

Prerequisites

- You installed RHEL 8.6 or later.
- You registered the system to the Customer Portal.
- You attached a **Red Hat Enterprise Linux Server** subscription to the system.
- If available in your Customer Portal account, you attached an **Ansible Automation Platform** subscription to the system.

Procedure

1. Install the **rhel-system-roles** package:

```
[root@control-node]# yum install rhel-system-roles
```

   This command installs **Ansible Core** as a dependency.

2. Create a user that you later use to manage and execute playbooks:

```
[root@control-node]# useradd ansible
```

3. Switch to the newly created **ansible** user:

```
[root@control-node]# su - ansible
```

   Perform the rest of the procedure as this user.

4. Create an SSH public and private key

```
[ansible@control-node]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ansible/.ssh/id_rsa): password
...
```

   Use the suggested default location for the key file.

5. Optional: Configure an SSH agent to prevent Ansible from prompting you for the SSH key password each time you establish a connection.

6. Create the ~/**.ansible.cfg** file with the following content:

```
[defaults]
inventory = /home/ansible/inventory
remote_user = ansible

[privilege_escalation]
become = True
become_method = sudo
become_user = root
become_ask_pass = True
```

   With these settings:

   - Ansible manages hosts in the specified inventory file.
   - Ansible uses the account set in the **remote_user** parameter when it establishes SSH connections to managed nodes.
   - Ansible uses the **sudo** utility to execute tasks on managed nodes as the **root** user. For security reasons, configure **sudo** on managed nodes to require entering the password of the remote user to become **root**. By specifying the **become_ask_pass=True** setting in ~/**.ansible.cfg**, Ansible prompts for this password when you execute a playbook.

   Settings in the ~/**.ansible.cfg** file have a higher priority and override settings from the global /**etc**/**ansible**/**ansible.cfg** file.

7. Create the ~/**inventory** file. For example, the following is an inventory file in the INI format with three hosts and one host group named **US**:

```
managed-node-01.example.com

[US]
managed-node-02.example.com ansible_host=192.0.2.100
managed-node-03.example.com
```

Note that the control node must be able to resolve the hostnames. If the DNS server cannot resolve certain hostnames, add the **ansible_host** parameter next to the host entry to specify its IP address.

Verification

1. Prepare a managed node.
2. Verify access from the control node to managed nodes

Additional resources

- Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories
- How to register and subscribe a system to the Red Hat Customer Portal using subscription-manager
- The **ssh-keygen(1)** man page
- Connecting to remote machines with SSH keys using ssh-agent
- Ansible configuration settings
- How to build your inventory
- Updates to using Ansible in RHEL 8.6 and 9.0

1.4. PREPARING A MANAGED NODE

Ansible does not use an agent on managed hosts. The only requirements are Python, which is installed by default on RHEL, and SSH access to the managed host.

However, direct SSH access as the **root** user can be a security risk. Therefore, when you prepare a managed node, you create a local user on this node and configure a **sudo** policy. Ansible on the control node can then use this account to log in to the managed node and execute playbooks as different users, such as **root**.

Prerequisites

- You prepared the control node.

Procedure

1. Create a user:

```
[root@managed-node-01]# useradd ansible
```

The control node later uses this user to establish an SSH connection to this host.

2. Set a password to the **ansible** user:

```
[root@managed-node-01]# passwd ansible
Changing password for user ansible.
New password: password
Retype new password: password
passwd: all authentication tokens updated successfully.
```

You must enter this password when Ansible uses **sudo** to perform tasks as the **root** user.

3. Install the **ansible** user's SSH public key on the managed node:

   a. Log into the control node as the **ansible** user, and copy the SSH public key to the managed node:

```
[ansible@control-node]$ ssh-copy-id managed-node-01.example.com
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed:
"/home/ansible/.ssh/id_rsa.pub"
The authenticity of host 'managed-node-01.example.com (192.0.2.100)' can't be
established.
ECDSA key fingerprint is
SHA256:9bZ33GJNODK3zbNhybokN/6Mq7hu3vpBXDrCxe7NAvo.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that
are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is
to install the new keys
ansible@managed-node-01.example.com's password: password

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'managed-node-01.example.com'"
and check to make sure that only the key(s) you wanted were added.
```

   b. Remotely execute a command on the control node to verify the SSH connection:

```
[ansible@control-node]$ ssh managed-node-01.example.com whoami
ansible
```

4. Create a **sudo** configuration for the **ansible** user:

   a. Use the **visudo** command to create and edit the **/etc/sudoers.d/ansible** file:

```
[root@managed-node-01]# visudo /etc/sudoers.d/ansible
```

   The benefit of using **visudo** over a normal editor is that this utility provides basic sanity checks and checks for parse errors before installing the file.

   b. Enter the following content to the **/etc/sudoers.d/ansible** file:

```
ansible ALL=(ALL) NOPASSWD: ALL
```

-

These settings grant permissions to the **ansible** user to run all commands as any user and group on this host without entering the password of the **ansible** user.

Additional resources

- Preparing the control node
- The **sudoers(5)** man page

1.5. VERIFYING ACCESS FROM THE CONTROL NODE TO MANAGED NODES

After you configured the control node and prepared managed nodes, test that Ansible can connect to the managed nodes.

Perform this procedure as the **ansible** user on the control node.

Prerequisites

- You prepared the control node as described in Preparing a control node.
- You prepared at least one managed node as described in Preparing a managed node.
- If you want to run playbooks on host groups, the managed node is listed in the inventory file on the control node.

Procedure

1. Use the Ansible **ping** module to verify that you can execute commands on an all managed hosts:

```
[ansible@control-node]$ ansible all -m ping
BECOME password: password
managed-node-01.example.com | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
...
```

   The hard-coded **all** host group dynamically contains all hosts listed in the inventory file.

2. Use the Ansible **command** module to run the **whoami** utility on a managed host:

```
[ansible@control-node]$ ansible managed-node-01.example.com -m command -a whoami
BECOME password: password
managed-node-01.example.com | CHANGED | rc=0 >>
root
```

   If the command returns **root**, you configured **sudo** on the managed nodes correctly, and privilege escalation works.

CHAPTER 2. UPDATING PACKAGES TO ENABLE AUTOMATION FOR RHEL SYSTEM ROLES

As of the RHEL 8.6 release, Ansible Engine is no longer supported. Instead, this and future RHEL versions include Ansible Core.

You can use Ansible Core in RHEL 8.6 to enable Ansible automation content written or generated by Red Hat products.

Ansible Core contains Ansible command line tools, such as the **ansible-playbook** and **ansible** commands, and a small set of built-in Ansible plugins .

2.1. DIFFERENCES BETWEEN ANSIBLE ENGINE AND ANSIBLE CORE

In RHEL 8.5 and earlier versions, you had access to a separate Ansible repository that contained Ansible Engine 2.9 to enable automation based on Ansible to your Red Hat system.

The scope of support, when using Ansible Engine without an Ansible subscription, is limited to running Ansible playbooks created or generated by Red Hat products, such as RHEL System Roles, Insights remediation playbooks, and OpenSCAP Ansible remediation playbooks.

In RHEL 8.6 and later versions, Ansible Core replaces Ansible Engine. The **ansible-core** package is included in the RHEL 9 AppStream repository to enable automation content provided by Red Hat. The scope of support for Ansible Core in RHEL remains the same as in earlier RHEL versions:

- Support is limited to any Ansible playbooks, roles, modules that are included with or generated by a Red Hat product, such as RHEL System Roles, or remediation playbooks generated by Insights.
- With Ansible Core, you get all functionality of supported RHEL Ansible content, such as RHEL System Roles and Insights remediation playbooks.

The Ansible Engine repository is still available in RHEL 8.6; however, it will not receive any security or bug fix updates and might not be compatible with Ansible automation content included in RHEL 8.6 and later.

You need an Ansible Automation Platform subscription for additional support for the underlying platform and Core-maintained modules.

Additional resources

- Scope of support for Ansible Core in RHEL

2.2. MIGRATING FROM ANSIBLE ENGINE TO ANSIBLE CORE

In RHEL 8.6 and later versions, Ansible Core replaces Ansible Engine. Migrate from Ansible Engine to Ansible Core to enable Ansible automation content generated by Red Hat products and get all needed functionality of supported RHEL Ansible content from RHEL 8.6 release.

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with RHEL System Roles.

- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Engine configures other systems.

On the control node:

- RHEL 8.6 or a later version is installed. You can verify your RHEL version by using the **cat /etc/redhat-release** command.
- The **rhel-system-roles** package is installed.
- An inventory file which lists the managed nodes.

Procedure

1. Uninstall Ansible Engine:

```
# yum remove ansible
```

2. Disable the **ansible-2-for-rhel-8-x86_64-rpms** repository:

```
# subscription-manager repos --disable ansible-2-for-rhel-8-x86_64-rpms
```

3. Install Ansible Core which is available in the RHEL 8 AppStream repository:

```
# yum install ansible-core
```

Verification

- Check that the **ansible-core** package is present in your system:

```
# yum info ansible-core
```

If the **ansible-core** package is indeed present in your system, the command output states information on the package name, version, release, size, and more:

```
Available Packages
Name         : ansible-core
Version      : 2.12.2
Release      : 1.fc34
Architecture : noarch
Size         : 2.4 M
Source       : ansible-core-2.12.2-1.fc34.src.rpm
Repository   : updates
Summary      : A radically simple IT automation system
URL          : http://ansible.com
```

CHAPTER 3. INSTALLING AND USING COLLECTIONS

3.1. INTRODUCTION TO ANSIBLE COLLECTIONS

Ansible Collections are the new way of distributing, maintaining, and consuming automation. By combining multiple types of Ansible content such as playbooks, roles, modules, and plugins, you can benefit from improvements in flexibility and scalability.

The Ansible Collections are an option to the traditional RHEL System Roles format. Using the RHEL System Roles in the Ansible Collection format is almost the same as using it in the traditional RHEL System Roles format. The difference is that Ansible Collections use the concept of a **fully qualified collection name** (FQCN), which consists of a **namespace** and the **collection name**. The **namespace** we use is **redhat** and the **collection name** is **rhel_system_roles**. So, while the traditional RHEL System Roles format for the Kernel Settings role is presented as **rhel-system-roles.kernel_settings**, using the Collection **fully qualified collection name** for the Kernel Settings role would be presented as **redhat.rhel_system_roles.kernel_settings**.

The combination of a **namespace** and a **collection name** guarantees that the objects are unique. It also ensures that objects are shared across the Ansible Collections and namespaces without any conflicts.

Additional resources

- To use the Red Hat Certified Collections by accessing the Automation Hub, you must have an Ansible Automation Platform (AAP subscription).

3.2. COLLECTIONS STRUCTURE

Collections are a package format for Ansible content. The data structure is as below:

- docs/: local documentation for the collection, with examples, if the role provides the documentation
- galaxy.yml: source data for the MANIFEST.json that will be part of the Ansible Collection package
- playbooks/: playbooks are available here
  - tasks/: this holds 'task list files' for include_tasks/import_tasks usage
- plugins/: all Ansible plugins and modules are available here, each in its subdirectory
  - modules/: Ansible modules
  - modules_utils/: common code for developing modules
  - lookup/: search for a plugin
  - filter/: Jinja2 filter plugin
  - connection/: connection plugins required if not using the default
- roles/: directory for Ansible roles
- tests/: tests for the collection's content

3.3. INSTALLING COLLECTIONS BY USING THE CLI

Collections are a distribution format for Ansible content that can include playbooks, roles, modules, and plugins.

You can install Collections through Ansible Galaxy, through the browser, or by using the command line.

Prerequisites

- Access and permissions to one or more *managed nodes*.
- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.
  On the control node:
  - The **ansible-core** and **rhel-system-roles** packages are installed.
  - An inventory file which lists the managed nodes.

Procedure

- Install the collection via RPM package:

```
# yum install rhel-system-roles
```

After the installation is finished, the roles are available as **redhat.rhel_system_roles.<role_name>**. Additionally, you can find the documentation for each role at **/usr/share/ansible/collections/ansible_collections/redhat/rhel_system_roles/roles/<role_name>/README.md**.

Verification steps

To verify that the Collections were successfully installed, you can apply the kernel_settings on your localhost:

1. Copy one of the **tests_default.yml** to your working directory.

```
$ cp
/usr/share/ansible/collections/ansible_collections/redhat/rhel_system_roles/tests/kernel_settings
ests_default.yml .
```

2. Edit the file, replacing "hosts: all" with "hosts: localhost" to make the playbook run only on the local system.
3. Run the ansible-playbook in the check mode. This does not change any settings on your system.

```
$ ansible-playbook --check tests_default.yml
```

The command returns the value **failed=0**.

Additional resources

- The **ansible-playbook** man page.

3.4. INSTALLING COLLECTIONS FROM AUTOMATION HUB

If you are using the Automation Hub, you can install the RHEL System Roles Collection hosted on the Automation Hub.

Prerequisites

- Access and permissions to one or more *managed nodes*.
- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.
  On the control node:
  - The **ansible-core** and **rhel-system-roles** packages are installed.
  - An inventory file which lists the managed nodes.

Procedure

1. Define Red Hat Automation Hub as the default source for content in the **ansible.cfg** configuration file. See Configuring Red Hat Automation Hub as the primary source for content .
2. Install the **redhat.rhel_system_roles** collection from the Automation Hub:

```
# ansible-galaxy collection install redhat.rhel_system_roles
```

   After the installation is finished, the roles are available as **redhat.rhel_system_roles.<role_name>**. Additionally, you can find the documentation for each role at **/usr/share/ansible/collections/ansible_collections/redhat/rhel_system_roles/roles/<role_name>/README.md**.

Verification steps

To verify that the Collections were successfully installed, you can apply the **kernel_settings** on your localhost:

1. Copy one of the **tests_default.yml** to your working directory.

```
$ cp
/usr/share/ansible/collections/ansible_collections/redhat/rhel_system_roles/tests/kernel_settings
ests_default.yml .
```

2. Edit the file, replacing "hosts: all" with "hosts: localhost" to make the playbook run only on the local system.
3. Run the ansible-playbook on the check mode. This does not change any settings on your system.

```
$ ansible-playbook --check tests_default.yml
```

   You can see the command returns with the value **failed=0**.

Additional resources

- The **ansible-playbook** man page.

3.5. APPLYING A LOCAL LOGGING SYSTEM ROLE USING COLLECTIONS

Following is an example using Collections to prepare and apply an Ansible playbook to configure a logging solution on a set of separate machines.

Prerequisites

- A Galaxy collection is installed.

Procedure

1. Create a playbook that defines the required role:
   a. Create a new YAML file and open it in a text editor, for example:

   ```
   # vi logging-playbook.yml
   ```

   b. Insert the following content into the YAML file:

   ```
   ---
   - name: Deploying basics input and implicit files output
     hosts: all
     roles:
       - redhat.rhel_system_roles.logging
     vars:
       logging_inputs:
         - name: system_input
           type: basics
       logging_outputs:
         - name: files_output
           type: files
       logging_flows:
         - name: flow1
           inputs: [system_input]
           outputs: [files_output]
   ```

2. Execute the playbook on a specific inventory:

   ```
   # ansible-playbook -i inventory-file logging-playbook.yml
   ```

   Where:

   - *inventory-file* is the name of your inventory file.
   - *logging-playbook.yml* is the playbook you use.

Verification steps

1. Test the syntax of the **/etc/rsyslog.conf** file:

   ```
   # rsyslogd -N 1
   rsyslogd: version 8.1911.0-6.el8, config validation run (level 1), master config
   /etc/rsyslog.conf
   ```

```
rsyslogd: End of config validation run. Bye.
```

2. Verify that the system sends messages to the log:

   a. Send a test message:

   ```
   # logger test
   ```

   b. View the **/var/log/messages** log, for example:

   ```
   # cat /var/log/messages
   Aug  5 13:48:31 hostname root[6778]: test
   ```

   The **hostname** is the hostname of the client system. The log displays the user name of the user that entered the logger command, in this case, **root**.

CHAPTER 4. ANSIBLE IPMI MODULES IN RHEL

4.1. THE RHEL_MGMT COLLECTION

The Intelligent Platform Management Interface (IPMI) is a specification for a set of standard protocols to communicate with baseboard management controller (BMC) devices. The **IPMI** modules allow you to enable and support hardware management automation. The **IPMI** modules are available in:

- The **rhel_mgmt** Collection. The package name is **ansible-collection-redhat-rhel_mgmt**.
- The RHEL 8 AppStream, as part of the new **ansible-collection-redhat-rhel_mgmt** package.

The following IPMI modules are available in the rhel_mgmt collection:

- **ipmi_boot**: Management of boot device order
- **ipmi_power**: Power management for machine

The mandatory parameters used for the IPMI Modules are:

- **ipmi_boot** parameters:

Module name	Description
name	Hostname or ip address of the BMC
password	Password to connect to the BMC
bootdev	Device to be used on next boot * network * floppy * hd * safe * optical * setup * default
User	Username to connect to the BMC

- **ipmi_power** parameters:

Module name	Description
name	BMC Hostname or IP address

Module name	Description
password	Password to connect to the BMC
user	Username to connect to the BMC
State	Check if the machine is on the desired status * on * off * shutdown * reset * boot

4.2. INSTALLING THE RHEL MGMT COLLECTION USING THE CLI

You can install the **rhel_mgmt** Collection using the command line.

Prerequisites

- The **ansible-core** package is installed.

Procedure

- Install the collection via RPM package:

```
# yum install ansible-collection-redhat-rhel_mgmt
```

After the installation is finished, the IPMI modules are available in the **redhat.rhel_mgmt** Ansible collection.

Additional resources

- The **ansible-playbook** man page.

4.3. EXAMPLE USING THE IPMI_BOOT MODULE

The following example shows how to use the **ipmi_boot** module in a playbook to set a boot device for the next boot. For simplicity, the examples use the same host as the Ansible control host and managed host, thus executing the modules on the same host where the playbook is executed.

Prerequisites

- The rhel_mgmt collection is installed.
- The **pyghmi** library in the **python3-pyghmi** package is installed in one of the following locations:
  - The host where you execute the playbook.

    - The managed host. If you use localhost as the managed host, install the **python3-pyghmi** package on the host where you execute the playbook instead.

- The IPMI BMC that you want to control is accessible via network from the host where you execute the playbook, or the managed host (if not using localhost as the managed host). Note that the host whose BMC is being configured by the module is generally different from the host where the module is executing (the Ansible managed host), as the module contacts the BMC over the network using the IPMI protocol.

- You have credentials to access BMC with an appropriate level of access.

Procedure

1. Create a new *playbook.yml* file with the following content:

```
---
- name: Sets which boot device will be used on next boot
  hosts: localhost
    tasks:
    - redhat.rhel_mgmt.ipmi_boot:
       name: bmc.host.example.com
         user: admin_user
         password: basics
         bootdev: hd
```

2. Execute the playbook against localhost:

```
# ansible-playbook playbook.yml
```

As a result, the output returns the value “success”.

4.4. EXAMPLE USING THE IPMI_POWER MODULE

This example shows how to use the **ipmi_boot** module in a playbook to check if the system is turned on. For simplicity, the examples use the same host as the Ansible control host and managed host, thus executing the modules on the same host where the playbook is executed.

Prerequisites

- The rhel_mgmt collection is installed.

- The **pyghmi** library in the **python3-pyghmi** package is installed in one of the following locations:

    - The host where you execute the playbook.

    - The managed host. If you use localhost as the managed host, install the **python3-pyghmi** package on the host where you execute the playbook instead.

- The IPMI BMC that you want to control is accessible via network from the host where you execute the playbook, or the managed host (if not using localhost as the managed host). Note that the host whose BMC is being configured by the module is generally different from the host where the module is executing (the Ansible managed host), as the module contacts the BMC over the network using the IPMI protocol.

- You have credentials to access BMC with an appropriate level of access.

Procedure

1. Create a new *playbook.yml* file with the following content:

```
---
- name: Turn the host on
  hosts: localhost
    tasks:
    - redhat.rhel_mgmt.ipmi_power:
        name: bmc.host.example.com
          user: admin_user
          password: basics
          state: on
```

2. Execute the playbook:

```
# ansible-playbook playbook.yml
```

The output returns the value “true”.

CHAPTER 5. THE REDFISH MODULES IN RHEL

The Redfish modules for remote management of devices are now part of the **redhat.rhel_mgmt** Ansible collection. With the Redfish modules, you can easily use management automation on bare-metal servers and platform hardware by getting information about the servers or control them through an Out-Of-Band (OOB) controller, using the standard HTTPS transport and JSON format.

5.1. THE REDFISH MODULES

The **redhat.rhel_mgmt** Ansible collection provides the Redfish modules to support hardware management in Ansible over Redfish. The **redhat.rhel_mgmt** collection is available in the **ansible-collection-redhat-rhel_mgmt** package. To install it, see Installing the redhat.rhel_mgmt Collection using the CLI.

The following Redfish modules are available in the **redhat.rhel_mgmt** collection:

1. **redfish_info**: The **redfish_info** module retrieves information about the remote Out-Of-Band (OOB) controller such as systems inventory.
2. **redfish_command**: The **redfish_command** module performs Out-Of-Band (OOB) controller operations like log management and user management, and power operations such as system restart, power on and off.
3. **redfish_config**: The **redfish_config** module performs OOB controller operations such as changing OOB configuration, or setting the BIOS configuration.

5.2. REDFISH MODULES PARAMETERS

The parameters used for the Redfish modules are:

redfish_info parameters:	Description
baseuri	(Mandatory) - Base URI of OOB controller.
category	(Mandatory) - List of categories to execute on OOB controller. The default value is ["Systems"].
command	(Mandatory) - List of commands to execute on OOB controller.
username	Username for authentication to OOB controller.
password	Password for authentication to OOB controller.

redfish_command parameters:	Description
baseuri	(Mandatory) - Base URI of OOB controller.
category	(Mandatory) - List of categories to execute on OOB controller. The default value is ["Systems"].

redfish_command parameters:	Description
command	(Mandatory) - List of commands to execute on OOB controller.
username	Username for authentication to OOB controller.
password	Password for authentication to OOB controller.

redfish_config parameters:	Description
baseuri	(Mandatory) - Base URI of OOB controller.
category	(Mandatory) - List of categories to execute on OOB controller. The default value is ["Systems"].
command	(Mandatory) - List of commands to execute on OOB controller.
username	Username for authentication to OOB controller.
password	Password for authentication to OOB controller.
bios_attributes	BIOS attributes to update.

5.3. USING THE REDFISH_INFO MODULE

The following example shows how to use the **redfish_info** module in a playbook to get information about the CPU inventory. For simplicity, the example uses the same host as the Ansible control host and managed host, thus executing the modules on the same host where the playbook is executed.

Prerequisites

- The **redhat.rhel_mgmt** collection is installed.
- The **pyghmi** library in the **python3-pyghmi** package is installed on the managed host. If you use localhost as the managed host, install the **python3-pyghmi** package on the host where you execute the playbook.
- OOB controller access details.

Procedure

1. Create a new *playbook.yml* file with the following content:

```
---
```

```
- name: Get CPU inventory
  hosts: localhost
  tasks:
    - redhat.rhel_mgmt.redfish_info:
        baseuri: "{{ baseuri }}"
        username: "{{ username }}"
        password: "{{ password }}"
        category: Systems
        command: GetCpuInventory
      register: result
```

2. Execute the playbook against localhost:

```
# ansible-playbook playbook.yml
```

As a result, the output returns the CPU inventory details.

5.4. USING THE REDFISH_COMMAND MODULE

The following example shows how to use the **redfish_command** module in a playbook to turn on a system. For simplicity, the example uses the same host as the Ansible control host and managed host, thus executing the modules on the same host where the playbook is executed.

Prerequisites

- The **redhat.rhel_mgmt** collection is installed.
- The **pyghmi** library in the **python3-pyghmi** package is installed on the managed host. If you use localhost as the managed host, install the **python3-pyghmi** package on the host where you execute the playbook.
- OOB controller access details.

Procedure

1. Create a new *playbook.yml* file with the following content:

```
---
- name: Power on system
  hosts: localhost
  tasks:
    - redhat.rhel_mgmt.redfish_command:
        baseuri: "{{ baseuri }}"
        username: "{{ username }}"
        password: "{{ password }}"
        category: Systems
        command: PowerOn
```

2. Execute the playbook against localhost:

```
# ansible-playbook playbook.yml
```

As a result, the system powers on.

5.5. USING THE REDFISH_CONFIG MODULE

The following example shows how to use the **redfish_config** module in a playbook to configure a system to boot with UEFI. For simplicity, the example uses the same host as the Ansible control host and managed host, thus executing the modules on the same host where the playbook is executed.

Prerequisites

- The **redhat.rhel_mgmt** collection is installed.
- The **pyghmi** library in the **python3-pyghmi** package is installed on the managed host. If you use localhost as the managed host, install the **python3-pyghmi** package on the host where you execute the playbook.
- OOB controller access details.

Procedure

1. Create a new *playbook.yml* file with the following content:

```
---
- name: "Set BootMode to UEFI"
  hosts: localhost
  tasks:
    - redhat.rhel_mgmt.redfish_config:
        baseuri: "{{ baseuri }}"
        username: "{{ username }}"
        password: "{{ password }}"
        category: Systems
        command: SetBiosAttributes
        bios_attributes:
          BootMode: Uefi
```

2. Execute the playbook against localhost:

```
# ansible-playbook playbook.yml
```

As a result, the system boot mode is set to UEFI.

CHAPTER 6. USING ANSIBLE ROLES TO PERMANENTLY CONFIGURE KERNEL PARAMETERS

As an experienced user with good knowledge of Red Hat Ansible, you can use the **kernel_settings** role to configure kernel parameters on multiple clients at once. This solution:

- Provides a friendly interface with efficient input setting.
- Keeps all intended kernel parameters in one place.

After you run the **kernel_settings** role from the control machine, the kernel parameters are applied to the managed systems immediately and persist across reboots.

IMPORTANT

Note that RHEL System Role delivered over RHEL channels are available to RHEL customers as an RPM package in the default AppStream repository. RHEL System Role are also available as a collection to customers with Ansible subscriptions over Ansible Automation Hub.

6.1. INTRODUCTION TO THE KERNEL SETTINGS ROLE

RHEL System Roles is a set of roles that provide a consistent configuration interface to remotely manage multiple systems.

RHEL System Roles were introduced for automated configurations of the kernel using the **kernel_settings** System Role. The **rhel-system-roles** package contains this system role, and also the reference documentation.

To apply the kernel parameters on one or more systems in an automated fashion, use the **kernel_settings** role with one or more of its role variables of your choice in a playbook. A playbook is a list of one or more plays that are human-readable, and are written in the YAML format.

You can use an inventory file to define a set of systems that you want Ansible to configure according to the playbook.

With the **kernel_settings** role you can configure:

- The kernel parameters using the **kernel_settings_sysctl** role variable
- Various kernel subsystems, hardware devices, and device drivers using the **kernel_settings_sysfs** role variable
- The CPU affinity for the **systemd** service manager and processes it forks using the **kernel_settings_systemd_cpu_affinity** role variable
- The kernel memory subsystem transparent hugepages using the **kernel_settings_transparent_hugepages** and **kernel_settings_transparent_hugepages_defrag** role variables

Additional resources

- **README.md** and **README.html** files in the **/usr/share/doc/rhel-system-roles/kernel_settings/** directory

- Working with playbooks
- How to build your inventory

6.2. APPLYING SELECTED KERNEL PARAMETERS USING THE KERNEL_SETTINGS ROLE

Follow these steps to prepare and apply an Ansible playbook to remotely configure kernel parameters with persisting effect on multiple managed operating systems.

Prerequisites

- You have **root** permissions.
- Entitled by your RHEL subscription, you installed the **ansible-core** and **rhel-system-roles** packages on the control machine.
- An inventory of managed hosts is present on the control machine and Ansible is able to connect to them.

IMPORTANT

RHEL 8.0 - 8.5 provided access to a separate Ansible repository that contains Ansible Engine 2.9 for automation based on Ansible. Ansible Engine contains command-line utilities such as **ansible**, **ansible-playbook**; connectors such as **docker** and **podman**; and the entire world of plugins and modules. For information on how to obtain and install Ansible Engine, refer to How do I Download and Install Red Hat Ansible Engine? .

RHEL 8.6 and 9.0 has introduced Ansible Core (provided as **ansible-core** RPM), which contains the Ansible command-line utilities, commands, and a small set of built-in Ansible plugins. The AppStream repository provides **ansible-core**, which has a limited scope of support. You can learn more by reviewing Scope of support for the ansible-core package included in the RHEL 9 AppStream.

Procedure

1. Optionally, review the **inventory** file for illustration purposes:

```
# cat /home/jdoe/<ansible_project_name>/inventory
[testingservers]
pdoe@192.168.122.98
fdoe@192.168.122.226

[db-servers]
db1.example.com
db2.example.com

[webservers]
web1.example.com
web2.example.com
192.0.2.42
```

The file defines the **[testingservers]** group and other groups. It allows you to run Ansible more effectively against a specific set of systems.

2. Create a configuration file to set defaults and privilege escalation for Ansible operations.

   a. Create a new YAML file and open it in a text editor, for example:

   ```
   # vi /home/jdoe/<ansible_project_name>/ansible.cfg
   ```

   b. Insert the following content into the file:

   ```
   [defaults]
   inventory = ./inventory

   [privilege_escalation]
   become = true
   become_method = sudo
   become_user = root
   become_ask_pass = true
   ```

   The **[defaults]** section specifies a path to the inventory file of managed hosts. The **[privilege_escalation]** section defines that user privileges be shifted to **root** on the specified managed hosts. This is necessary for successful configuration of kernel parameters. When Ansible playbook is run, you will be prompted for user password. The user automatically switches to **root** by means of **sudo** after connecting to a managed host.

3. Create an Ansible playbook that uses the **kernel_settings** role.

   a. Create a new YAML file and open it in a text editor, for example:

   ```
   # vi /home/jdoe/<ansible_project_name>/kernel-roles.yml
   ```

   This file represents a playbook and usually contains an ordered list of tasks, also called *plays*, that are run against specific managed hosts selected from your **inventory** file.

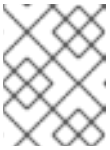
   b. Insert the following content into the file:

   ```
   ---
   -
     hosts: testingservers
     name: "Configure kernel settings"
     roles:
       - rhel-system-roles.kernel_settings
     vars:
       kernel_settings_sysctl:
         - name: fs.file-max
           value: 400000
         - name: kernel.threads-max
           value: 65536
       kernel_settings_sysfs:
         - name: /sys/class/net/lo/mtu
           value: 65000
       kernel_settings_transparent_hugepages: madvise
   ```

   The **name** key is optional. It associates an arbitrary string with the play as a label and identifies what the play is for. The **hosts** key in the play specifies the hosts against which the play is run. The value or values for this key can be provided as individual names of managed hosts or as groups of hosts as defined in the **inventory** file.

The **vars** section represents a list of variables containing selected kernel parameter names and values to which they have to be set.

The **roles** key specifies what system role is going to configure the parameters and values mentioned in the **vars** section.



NOTE

You can modify the kernel parameters and their values in the playbook to fit your needs.

4. Optionally, verify that the syntax in your play is correct.

```
# ansible-playbook --syntax-check kernel-roles.yml

playbook: kernel-roles.yml
```

This example shows the successful verification of a playbook.

5. Execute your playbook.

```
# ansible-playbook kernel-roles.yml

...

BECOME password:

PLAY [Configure kernel settings]
*********************************************************************************



PLAY RECAP
*********************************************************************************************************
fdoe@192.168.122.226      : ok=10   changed=4    unreachable=0    failed=0    skipped=6
rescued=0    ignored=0
pdoe@192.168.122.98       : ok=10   changed=4    unreachable=0    failed=0    skipped=6
rescued=0    ignored=0
```

Before Ansible runs your playbook, you are going to be prompted for your password and so that a user on managed hosts can be switched to **root**, which is necessary for configuring kernel parameters.

The recap section shows that the play finished successfully (**failed=0**) for all managed hosts, and that 4 kernel parameters have been applied (**changed=4**).

6. Restart your managed hosts and check the affected kernel parameters to verify that the changes have been applied and persist across reboots.

Additional resources

- Preparing a control node and managed nodes to use RHEL System Roles
- **README.html** and **README.md** files in the **/usr/share/doc/rhel-system-roles/kernel_settings/** directory

- Build Your Inventory
- Configuring Ansible
- Working With Playbooks
- Using Variables
- Roles

CHAPTER 7. USING THE NETWORK SYSTEM ROLE TO CONFIGURE THE NETWORK

The **network** RHEL System Role enables administrators to automate network-related configuration and management tasks using Ansible.

7.1. CONFIGURING A STATIC ETHERNET CONNECTION USING RHEL SYSTEM ROLES WITH THE INTERFACE NAME

This procedure describes how to use the **network** RHEL System Role to remotely add an Ethernet connection for the **enp7s0** interface with the following settings by running an Ansible playbook:

- A static IPv4 address - **192.0.2.1** with a /**24** subnet mask
- A static IPv6 address - **2001:db8:1::1** with a /**64** subnet mask
- An IPv4 default gateway - **192.0.2.254**
- An IPv6 default gateway - **2001:db8:1::fffe**
- An IPv4 DNS server - **192.0.2.200**
- An IPv6 DNS server - **2001:db8:1::ffbb**
- A DNS search domain - **example.com**

Perform this procedure on the Ansible control node.

Prerequisites

- You have prepared the control node and the managed nodes .
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.
- The hosts or host groups on which you to want run this playbook are listed in the Ansible inventory file.
- The managed nodes use NetworkManager to configure the network.

Procedure

1. Create a playbook file, for example ***~/ethernet-static-IP.yml***, with the following content:

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
  - name: Configure an Ethernet connection with static IP
    include_role:
      name: rhel-system-roles.network

    vars:
      network_connections:
```

```
    - name: enp7s0
      interface_name: enp7s0
      type: ethernet
      autoconnect: yes
      ip:
        address:
          - 192.0.2.1/24
          - 2001:db8:1::1/64
        gateway4: 192.0.2.254
        gateway6: 2001:db8:1::fffe
        dns:
          - 192.0.2.200
          - 2001:db8:1::ffbb
        dns_search:
          - example.com
      state: up
```

2. Run the playbook:

```
# ansible-playbook ~/ethernet-static-IP.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md**

7.2. CONFIGURING A STATIC ETHERNET CONNECTION USING RHEL SYSTEM ROLES WITH A DEVICE PATH

This procedure describes how to use RHEL System Roles to remotely add an Ethernet connection with static IP address for devices that match a specific device path by running an Ansible playbook.

You can identify the device path with the following command:

```
# udevadm info /sys/class/net/<device_name> | grep ID_PATH=
```

This procedure sets the following settings to the device that matches the PCI ID **0000:00:0[1-3].0** expression, but not **0000:00:02.0**:

- A static IPv4 address - **192.0.2.1** with a **/24** subnet mask
- A static IPv6 address - **2001:db8:1::1** with a **/64** subnet mask
- An IPv4 default gateway - **192.0.2.254**
- An IPv6 default gateway - **2001:db8:1::fffe**
- An IPv4 DNS server - **192.0.2.200**
- An IPv6 DNS server - **2001:db8:1::ffbb**
- A DNS search domain - **example.com**

Perform this procedure on the Ansible control node.

Prerequisites

- You have prepared the control node and the managed nodes .
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.
- The hosts or host groups on which you want to run this playbook are listed in the Ansible inventory file.
- The managed nodes use NetworkManager to configure the network.

Procedure

1. Create a playbook file, for example ***~/ethernet-static-IP.yml***, with the following content:

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
  - name: Configure an Ethernet connection with dynamic IP
    include_role:
      name: rhel-system-roles.network

    vars:
      network_connections:
        - name: example
          match:
            path:
              - pci-0000:00:0[1-3].0
              - &!pci-0000:00:02.0
          type: ethernet
          autoconnect: yes
          ip:
            address:
              - 192.0.2.1/24
              - 2001:db8:1::1/64
            gateway4: 192.0.2.254
            gateway6: 2001:db8:1::fffe
            dns:
              - 192.0.2.200
              - 2001:db8:1::ffbb
            dns_search:
              - example.com
          state: up
```

   The **match** parameter in this example defines that Ansible applies the play to devices that match PCI ID **0000:00:0[1-3].0**, but not **0000:00:02.0**. For further details about special modifiers and wild cards you can use, see the **match** parameter description in the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file.

2. Run the playbook:

```
# ansible-playbook ~/ethernet-static-IP.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file

7.3. CONFIGURING A DYNAMIC ETHERNET CONNECTION USING RHEL SYSTEM ROLES WITH THE INTERFACE NAME

This procedure describes how to use RHEL System Roles to remotely add a dynamic Ethernet connection for the **enp7s0** interface by running an Ansible playbook. With this setting, the network connection requests the IP settings for this connection from a DHCP server.

Perform this procedure on the Ansible control node.

Prerequisites

- You have prepared the control node and the managed nodes .
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.
- The hosts or host groups on which you want to run this playbook are listed in the Ansible inventory file.
- A DHCP server is available in the network
- The managed nodes use NetworkManager to configure the network.

Procedure

1. Create a playbook file, for example ***~/ethernet-dynamic-IP.yml***, with the following content:

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
  - name: Configure an Ethernet connection with dynamic IP
    include_role:
      name: rhel-system-roles.network

    vars:
      network_connections:
        - name: enp7s0
          interface_name: enp7s0
          type: ethernet
          autoconnect: yes
          ip:
            dhcp4: yes
            auto6: yes
          state: up
```

2. Run the playbook:

```
# ansible-playbook ~/ethernet-dynamic-IP.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file

7.4. CONFIGURING A DYNAMIC ETHERNET CONNECTION USING RHEL SYSTEM ROLES WITH A DEVICE PATH

This procedure describes how to use RHEL System Roles to remotely add a dynamic Ethernet connection for devices that match a specific device path by running an Ansible playbook. With dynamic IP settings, the network connection requests the IP settings for this connection from a DHCP server.

You can identify the device path with the following command:

```
# udevadm info /sys/class/net/<device_name> | grep ID_PATH=
```

Perform this procedure on the Ansible control node.

Prerequisites

- You have prepared the control node and the managed nodes .
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.
- The hosts or host groups on which you want to run this playbook are listed in the Ansible inventory file.
- A DHCP server is available in the network.
- The managed hosts use NetworkManager to configure the network.

Procedure

1. Create a playbook file, for example ***~/ethernet-dynamic-IP.yml***, with the following content:

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
  - name: Configure an Ethernet connection with dynamic IP
    include_role:
      name: rhel-system-roles.network

    vars:
      network_connections:
        - name: example
          match:
            path:
              - pci-0000:00:0[1-3].0
              - &!pci-0000:00:02.0
          type: ethernet
          autoconnect: yes
          ip:
```

```
      dhcp4: yes
      auto6: yes
    state: up
```

The **match** parameter in this example defines that Ansible applies the play to devices that match PCI ID **0000:00:0[1-3].0**, but not **0000:00:02.0**. For further details about special modifiers and wild cards you can use, see the **match** parameter description in the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file.

2. Run the playbook:

```
# ansible-playbook ~/ethernet-dynamic-IP.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file

7.5. CONFIGURING VLAN TAGGING USING RHEL SYSTEM ROLES

You can use the **network** RHEL System Role to configure VLAN tagging. This example adds an Ethernet connection and a VLAN with ID **10** on top of this Ethernet connection. As the child device, the VLAN connection contains the IP, default gateway, and DNS configurations.

Depending on your environment, adjust the play accordingly. For example:

- To use the VLAN as a port in other connections, such as a bond, omit the **ip** attribute, and set the IP configuration in the child configuration.
- To use team, bridge, or bond devices in the VLAN, adapt the **interface_name** and **type** attributes of the ports you use in the VLAN.

Perform this procedure on the Ansible control node.

Prerequisites

- You have prepared the control node and the managed nodes .
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.
- The hosts or host groups on which you to want run this playbook are listed in the Ansible inventory file.

Procedure

1. Create a playbook file, for example ***~/vlan-ethernet.yml***, with the following content:

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
  - name: Configure a VLAN that uses an Ethernet connection
    include_role:
      name: rhel-system-roles.network
```

```
vars:
  network_connections:
    # Add an Ethernet profile for the underlying device of the VLAN
    - name: enp1s0
      type: ethernet
      interface_name: enp1s0
      autoconnect: yes
      state: up
      ip:
        dhcp4: no
        auto6: no

    # Define the VLAN profile
    - name: enp1s0.10
      type: vlan
      ip:
        address:
          - "192.0.2.1/24"
          - "2001:db8:1::1/64"
        gateway4: 192.0.2.254
        gateway6: 2001:db8:1::fffe
        dns:
          - 192.0.2.200
          - 2001:db8:1::ffbb
        dns_search:
          - example.com
      vlan_id: 10
      parent: enp1s0
      state: up
```

The **parent** attribute in the VLAN profile configures the VLAN to operate on top of the **enp1s0** device.

2. Run the playbook:

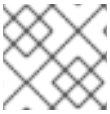
```
# ansible-playbook ~/vlan-ethernet.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file

7.6. CONFIGURING A NETWORK BRIDGE USING RHEL SYSTEM ROLES

You can use the **network** RHEL System Role to configure a Linux bridge. This procedure describes how to configure a network bridge that uses two Ethernet devices, and sets IPv4 and IPv6 addresses, default gateways, and DNS configuration.



NOTE

Set the IP configuration on the bridge and not on the ports of the Linux bridge.

Perform this procedure on the Ansible control node.

Prerequisites

- You have prepared the control node and the managed nodes .
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.
- The hosts or host groups on which you want to run this playbook are listed in the Ansible inventory file.
- Two or more physical or virtual network devices are installed on the server.

Procedure

1. Create a playbook file, for example ***~/bridge-ethernet.yml***, with the following content:

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
  - name: Configure a network bridge that uses two Ethernet ports
    include_role:
      name: rhel-system-roles.network

    vars:
      network_connections:
        # Define the bridge profile
        - name: bridge0
          type: bridge
          interface_name: bridge0
          ip:
            address:
              - "192.0.2.1/24"
              - "2001:db8:1::1/64"
            gateway4: 192.0.2.254
            gateway6: 2001:db8:1::fffe
            dns:
              - 192.0.2.200
              - 2001:db8:1::ffbb
            dns_search:
              - example.com
          state: up

        # Add an Ethernet profile to the bridge
        - name: bridge0-port1
          interface_name: enp7s0
          type: ethernet
          controller: bridge0
          port_type: bridge
          state: up

        # Add a second Ethernet profile to the bridge
        - name: bridge0-port2
          interface_name: enp8s0
          type: ethernet
```

```
      controller: bridge0
      port_type: bridge
      state: up
```

2. Run the playbook:

```
# ansible-playbook ~/bridge-ethernet.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file

7.7. CONFIGURING A NETWORK BOND USING RHEL SYSTEM ROLES

You can use the **network** RHEL System Role to configure a network bond. This procedure describes how to configure a bond in active-backup mode that uses two Ethernet devices, and sets an IPv4 and IPv6 addresses, default gateways, and DNS configuration.

NOTE

Set the IP configuration on the bond and not on the ports of the Linux bond.

Perform this procedure on the Ansible control node.

Prerequisites

- You have prepared the control node and the managed nodes .
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.
- The hosts or host groups on which you want to run this playbook are listed in the Ansible inventory file.
- Two or more physical or virtual network devices are installed on the server.

Procedure

1. Create a playbook file, for example *~/**bond-ethernet.yml***, with the following content:

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
  name: Configure a network bond that uses two Ethernet ports
  - include_role:
      name: rhel-system-roles.network

    vars:
      network_connections:
        # Define the bond profile
        - name: bond0
          type: bond
```

```
    interface_name: bond0
    ip:
      address:
        - "192.0.2.1/24"
        - "2001:db8:1::1/64"
      gateway4: 192.0.2.254
      gateway6: 2001:db8:1::fffe
      dns:
        - 192.0.2.200
        - 2001:db8:1::ffbb
      dns_search:
        - example.com
    bond:
      mode: active-backup
    state: up

  # Add an Ethernet profile to the bond
  - name: bond0-port1
    interface_name: enp7s0
    type: ethernet
    controller: bond0
    state: up

  # Add a second Ethernet profile to the bond
  - name: bond0-port2
    interface_name: enp8s0
    type: ethernet
    controller: bond0
    state: up
```

2. Run the playbook:

```
# ansible-playbook ~/bond-ethernet.yml
```

Additional resources

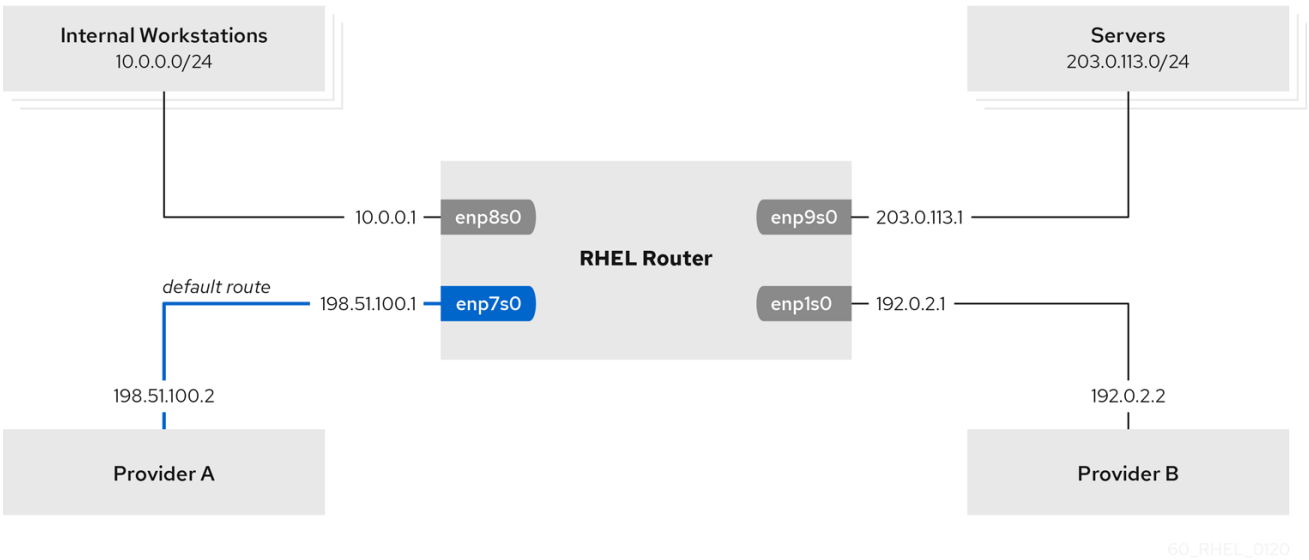
- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file

7.8. ROUTING TRAFFIC FROM A SPECIFIC SUBNET TO A DIFFERENT DEFAULT GATEWAY USING RHEL SYSTEM ROLES

You can use policy-based routing to configure a different default gateway for traffic from certain subnets. For example, you can configure RHEL as a router that, by default, routes all traffic to Internet provider A using the default route. However, traffic received from the internal workstations subnet is routed to provider B.

To configure policy-based routing remotely and on multiple nodes, you can use the RHEL **network** System Role. Perform this procedure on the Ansible control node.

This procedure assumes the following network topology:



Prerequisites

- You have prepared the control node and the managed nodes .
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on the them.
- The hosts or host groups on which you want run this playbook are listed in the Ansible inventory file.
- The managed nodes uses the **NetworkManager** and **firewalld** services.
- The managed nodes you want to configure has four network interfaces:
  - The **enp7s0** interface is connected to the network of provider A. The gateway IP in the provider's network is **198.51.100.2**, and the network uses a **/30** network mask.
  - The **enp1s0** interface is connected to the network of provider B. The gateway IP in the provider's network is **192.0.2.2**, and the network uses a **/30** network mask.
  - The **enp8s0** interface is connected to the **10.0.0.0/24** subnet with internal workstations.
  - The **enp9s0** interface is connected to the **203.0.113.0/24** subnet with the company's servers.
- Hosts in the internal workstations subnet use **10.0.0.1** as the default gateway. In the procedure, you assign this IP address to the **enp8s0** network interface of the router.
- Hosts in the server subnet use **203.0.113.1** as the default gateway. In the procedure, you assign this IP address to the **enp9s0** network interface of the router.

Procedure

1. Create a playbook file, for example ~/**pbr.yml**, with the following content:

```
---
- name: Configuring policy-based routing
```

```
hosts: managed-node-01.example.com
tasks:
- name: Routing traffic from a specific subnet to a different default gateway
  include_role:
    name: rhel-system-roles.network

  vars:
    network_connections:
      - name: Provider-A
        interface_name: enp7s0
        type: ethernet
        autoconnect: True
        ip:
          address:
            - 198.51.100.1/30
          gateway4: 198.51.100.2
          dns:
            - 198.51.100.200
        state: up
        zone: external

      - name: Provider-B
        interface_name: enp1s0
        type: ethernet
        autoconnect: True
        ip:
          address:
            - 192.0.2.1/30
          route:
            - network: 0.0.0.0
              prefix: 0
              gateway: 192.0.2.2
              table: 5000
        state: up
        zone: external

      - name: Internal-Workstations
        interface_name: enp8s0
        type: ethernet
        autoconnect: True
        ip:
          address:
            - 10.0.0.1/24
          route:
            - network: 10.0.0.0
              prefix: 24
              table: 5000
          routing_rule:
            - priority: 5
              from: 10.0.0.0/24
              table: 5000
        state: up
        zone: trusted

      - name: Servers
        interface_name: enp9s0
```

```
      type: ethernet
      autoconnect: True
      ip:
        address:
          - 203.0.113.1/24
      state: up
      zone: trusted
```

2. Run the playbook:

```
# ansible-playbook ~/pbr.yml
```

Verification

1. On a RHEL host in the internal workstation subnet:
   a. Install the **traceroute** package:

```
# yum install traceroute
```

   b. Use the **traceroute** utility to display the route to a host on the Internet:

```
# traceroute redhat.com
traceroute to redhat.com (209.132.183.105), 30 hops max, 60 byte packets
 1  10.0.0.1 (10.0.0.1)     0.337 ms  0.260 ms  0.223 ms
 2  192.0.2.1 (192.0.2.1)   0.884 ms  1.066 ms  1.248 ms
 ...
```

   The output of the command displays that the router sends packets over **192.0.2.1**, which is the network of provider B.

2. On a RHEL host in the server subnet:
   a. Install the **traceroute** package:

```
# yum install traceroute
```

   b. Use the **traceroute** utility to display the route to a host on the Internet:

```
# traceroute redhat.com
traceroute to redhat.com (209.132.183.105), 30 hops max, 60 byte packets
 1  203.0.113.1 (203.0.113.1)    2.179 ms  2.073 ms  1.944 ms
 2  198.51.100.2 (198.51.100.2)  1.868 ms  1.798 ms  1.549 ms
 ...
```

   The output of the command displays that the router sends packets over **198.51.100.2**, which is the network of provider A.

3. On the RHEL router that you configured using the RHEL System Role:
   a. Display the rule list:

```
# ip rule list
0:      from all lookup local
```

```
5:    from 10.0.0.0/24 lookup 5000
32766:  from all lookup main
32767:  from all lookup default
```

By default, RHEL contains rules for the tables **local**, **main**, and **default**.

b. Display the routes in table **5000**:

```
# ip route list table 5000
0.0.0.0/0 via 192.0.2.2 dev enp1s0 proto static metric 100
10.0.0.0/24 dev enp8s0 proto static scope link src 192.0.2.1 metric 102
```

c. Display the interfaces and firewall zones:

```
# firewall-cmd --get-active-zones
external
  interfaces: enp1s0 enp7s0
trusted
  interfaces: enp8s0 enp9s0
```

d. Verify that the **external** zone has masquerading enabled:

```
# firewall-cmd --info-zone=external
external (active)
  target: default
  icmp-block-inversion: no
  interfaces: enp1s0 enp7s0
  sources:
  services: ssh
  ports:
  protocols:
  masquerade: yes
  ...
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md**

7.9. CONFIGURING A STATIC ETHERNET CONNECTION WITH 802.1X NETWORK AUTHENTICATION USING RHEL SYSTEM ROLES

Using the **network** RHEL System Role, you can automate the creation of an Ethernet connection that uses the 802.1X standard to authenticate the client. This procedure describes how to remotely add an Ethernet connection for the **enp1s0** interface with the following settings by running an Ansible playbook:

- A static IPv4 address - **192.0.2.1** with a **/24** subnet mask
- A static IPv6 address - **2001:db8:1::1** with a **/64** subnet mask
- An IPv4 default gateway - **192.0.2.254**
- An IPv6 default gateway - **2001:db8:1::fffe**

- An IPv4 DNS server - **192.0.2.200**
- An IPv6 DNS server - **2001:db8:1::ffbb**
- A DNS search domain - **example.com**
- 802.1X network authentication using the **TLS** Extensible Authentication Protocol (EAP)

Perform this procedure on the Ansible control node.

Prerequisites

- You have prepared the control node and the managed nodes .
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.
- The hosts or host groups on which you want to run this playbook are listed in the Ansible inventory file.
- The network supports 802.1X network authentication.
- The managed nodes uses NetworkManager.
- The following files required for TLS authentication exist on the control node:
  - The client key is stored in the **/srv/data/client.key** file.
  - The client certificate is stored in the **/srv/data/client.crt** file.
  - The Certificate Authority (CA) certificate is stored in the **/srv/data/ca.crt** file.

Procedure

1. Create a playbook file, for example ***~/enable-802.1x.yml***, with the following content:

```
---
- name: Configure an Ethernet connection with 802.1X authentication
  hosts: managed-node-01.example.com
  tasks:
    - name: Copy client key for 802.1X authentication
      copy:
        src: "/srv/data/client.key"
        dest: "/etc/pki/tls/private/client.key"
        mode: 0600

    - name: Copy client certificate for 802.1X authentication
      copy:
        src: "/srv/data/client.crt"
        dest: "/etc/pki/tls/certs/client.crt"

    - name: Copy CA certificate for 802.1X authentication
      copy:
        src: "/srv/data/ca.crt"
        dest: "/etc/pki/ca-trust/source/anchors/ca.crt"
```

```
- include_role:
    name: rhel-system-roles.network

  vars:
    network_connections:
      - name: enp1s0
        type: ethernet
        autoconnect: yes
        ip:
          address:
            - 192.0.2.1/24
            - 2001:db8:1::1/64
          gateway4: 192.0.2.254
          gateway6: 2001:db8:1::fffe
          dns:
            - 192.0.2.200
            - 2001:db8:1::ffbb
          dns_search:
            - example.com
        ieee802_1x:
          identity: user_name
          eap: tls
          private_key: "/etc/pki/tls/private/client.key"
          private_key_password: "password"
          client_cert: "/etc/pki/tls/certs/client.crt"
          ca_cert: "/etc/pki/ca-trust/source/anchors/ca.crt"
          domain_suffix_match: example.com
        state: up
```

2. Run the playbook:

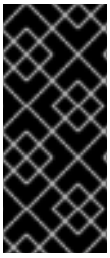
```
# ansible-playbook ~/enable-802.1x.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file

7.10. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION USING RHEL SYSTEM ROLES

You can use the **network** RHEL System Role to set the default gateway.



IMPORTANT

When you run a play that uses the **network** RHEL System Role, the system role overrides an existing connection profile with the same name if the value of settings does not match the ones specified in the play. Therefore, always specify the whole configuration of the network connection profile in the play, even if, for example, the IP configuration already exists. Otherwise, the role resets these values to their defaults.

Depending on whether it already exists, the procedure creates or updates the **enp1s0** connection profile with the following settings:

- A static IPv4 address - **198.51.100.20** with a **/24** subnet mask

- A static IPv6 address - **2001:db8:1::1** with a **/64** subnet mask
- An IPv4 default gateway - **198.51.100.254**
- An IPv6 default gateway - **2001:db8:1::fffe**
- An IPv4 DNS server - **198.51.100.200**
- An IPv6 DNS server - **2001:db8:1::ffbb**
- A DNS search domain - **example.com**

Perform this procedure on the Ansible control node.

Prerequisites

- You have prepared the control node and the managed nodes .
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.
- The hosts or host groups on which you want to run this playbook are listed in the Ansible inventory file.

Procedure

1. Create a playbook file, for example ***~/ethernet-connection.yml***, with the following content:

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
  - name: Configure an Ethernet connection with static IP and default gateway
    include_role:
      name: rhel-system-roles.network

    vars:
      network_connections:
        - name: enp1s0
          type: ethernet
          autoconnect: yes
          ip:
            address:
              - 198.51.100.20/24
              - 2001:db8:1::1/64
            gateway4: 198.51.100.254
            gateway6: 2001:db8:1::fffe
            dns:
              - 198.51.100.200
              - 2001:db8:1::ffbb
            dns_search:
              - example.com
          state: up
```

2. Run the playbook:

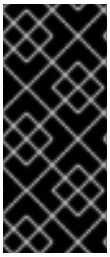
```
# ansible-playbook ~/ethernet-connection.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md**

7.11. CONFIGURING A STATIC ROUTE USING RHEL SYSTEM ROLES

You can use the **network** RHEL System Role to configure static routes.



IMPORTANT

When you run a play that uses the **network** RHEL System Role, the system role overrides an existing connection profile with the same name if the value of settings does not match the ones specified in the play. Therefore, always specify the whole configuration of the network connection profile in the play, even if, for example, the IP configuration already exists. Otherwise, the role resets these values to their defaults.

Depending on whether it already exists, the procedure creates or updates the **enp7s0** connection profile with the following settings:

- A static IPv4 address - **192.0.2.1** with a **/24** subnet mask
- A static IPv6 address - **2001:db8:1::1** with a **/64** subnet mask
- An IPv4 default gateway - **192.0.2.254**
- An IPv6 default gateway - **2001:db8:1::fffe**
- An IPv4 DNS server - **192.0.2.200**
- An IPv6 DNS server - **2001:db8:1::ffbb**
- A DNS search domain - **example.com**
- Static routes:
  - **198.51.100.0/24** with gateway **192.0.2.10**
  - **2001:db8:2::/64** with gateway **2001:db8:1::10**

Perform this procedure on the Ansible control node.

Prerequisites

- You have prepared the control node and the managed nodes .
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.
- The hosts or host groups on which you to want run this playbook are listed in the Ansible inventory file.

Procedure

1. Create a playbook file, for example *~/**add-static-routes.yml***, with the following content:

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
  - name: Configure an Ethernet connection with static IP and additional routes
    include_role:
      name: rhel-system-roles.network

    vars:
      network_connections:
        - name: enp7s0
          type: ethernet
          autoconnect: yes
          ip:
            address:
              - 192.0.2.1/24
              - 2001:db8:1::1/64
            gateway4: 192.0.2.254
            gateway6: 2001:db8:1::fffe
            dns:
              - 192.0.2.200
              - 2001:db8:1::ffbb
            dns_search:
              - example.com
            route:
              - network: 198.51.100.0
                prefix: 24
                gateway: 192.0.2.10
              - network: 2001:db8:2::
                prefix: 64
                gateway: 2001:db8:1::10
          state: up
```

2. Run the playbook:

```
# ansible-playbook ~/add-static-routes.yml
```

Verification steps

1. On the managed nodes:

   a. Display the IPv4 routes:

```
# ip -4 route
...
198.51.100.0/24 via 192.0.2.10 dev enp7s0
```

   b. Display the IPv6 routes:

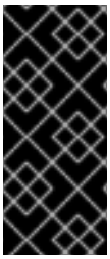
```
# ip -6 route
...
2001:db8:2::/64 via 2001:db8:1::10 dev enp7s0 metric 1024 pref medium
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md**

7.12. USING RHEL SYSTEM ROLES TO SET ETHTOOL FEATURES

You can use the **network** RHEL System Role to configure **ethtool** features of a NetworkManager connection.



IMPORTANT

When you run a play that uses the **network** RHEL System Role, the system role overrides an existing connection profile with the same name if the value of settings does not match the ones specified in the play. Therefore, always specify the whole configuration of the network connection profile in the play, even if, for example the IP configuration, already exists. Otherwise the role resets these values to their defaults.

Depending on whether it already exists, the procedure creates or updates the **enp1s0** connection profile with the following settings:

- A static **IPv4** address - **198.51.100.20** with a **/24** subnet mask
- A static **IPv6** address - **2001:db8:1::1** with a **/64** subnet mask
- An **IPv4** default gateway - **198.51.100.254**
- An **IPv6** default gateway - **2001:db8:1::fffe**
- An **IPv4** DNS server - **198.51.100.200**
- An **IPv6** DNS server - **2001:db8:1::ffbb**
- A DNS search domain - **example.com**
- **ethtool** features:
  - Generic receive offload (GRO): disabled
  - Generic segmentation offload (GSO): enabled
  - TX stream control transmission protocol (SCTP) segmentation: disabled

Perform this procedure on the Ansible control node.

Prerequisites

- You have prepared the control node and the managed nodes .
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.

- The hosts or host groups on which you want to run this playbook are listed in the Ansible inventory file.

Procedure

1. Create a playbook file, for example ***~/configure-ethernet-device-with-ethtool-features.yml***, with the following content:

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
  - name: Configure an Ethernet connection with ethtool features
    include_role:
      name: rhel-system-roles.network

    vars:
      network_connections:
        - name: enp1s0
          type: ethernet
          autoconnect: yes
          ip:
            address:
              - 198.51.100.20/24
              - 2001:db8:1::1/64
            gateway4: 198.51.100.254
            gateway6: 2001:db8:1::fffe
            dns:
              - 198.51.100.200
              - 2001:db8:1::ffbb
            dns_search:
              - example.com
          ethtool:
            features:
              gro: "no"
              gso: "yes"
              tx_sctp_segmentation: "no"
          state: up
```

2. Run the playbook:

```
# ansible-playbook ~/configure-ethernet-device-with-ethtool-features.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file

7.13. USING RHEL SYSTEM ROLES TO CONFIGURE ETHTOOL COALESCE SETTINGS

You can use the **network** RHEL System Role to configure **ethtool** coalesce settings of a NetworkManager connection.

IMPORTANT

When you run a play that uses the **network** RHEL System Role, the system role overrides an existing connection profile with the same name if the value of settings does not match the ones specified in the play. Therefore, always specify the whole configuration of the network connection profile in the play, even if, for example the IP configuration, already exists. Otherwise the role resets these values to their defaults.

Depending on whether it already exists, the procedure creates or updates the **enp1s0** connection profile with the following settings:

- A static IPv4 address - **198.51.100.20** with a **/24** subnet mask
- A static IPv6 address - **2001:db8:1::1** with a **/64** subnet mask
- An IPv4 default gateway - **198.51.100.254**
- An IPv6 default gateway - **2001:db8:1::fffe**
- An IPv4 DNS server - **198.51.100.200**
- An IPv6 DNS server - **2001:db8:1::ffbb**
- A DNS search domain - **example.com**
- **ethtool** coalesce settings:
  - RX frames: **128**
  - TX frames: **128**

Perform this procedure on the Ansible control node.

Prerequisites

- You have prepared the control node and the managed nodes .
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.
- The hosts or host groups on which you want to run this playbook are listed in the Ansible inventory file.

Procedure

1. Create a playbook file, for example ***~/configure-ethernet-device-with-ethtoolcoalesce-settings.yml***, with the following content:

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
  - name: Configure an Ethernet connection with ethtool coalesce settings
    include_role:
      name: rhel-system-roles.network
```

```
vars:
  network_connections:
    - name: enp1s0
      type: ethernet
      autoconnect: yes
      ip:
        address:
          - 198.51.100.20/24
          - 2001:db8:1::1/64
        gateway4: 198.51.100.254
        gateway6: 2001:db8:1::fffe
        dns:
          - 198.51.100.200
          - 2001:db8:1::ffbb
        dns_search:
          - example.com
      ethtool:
        coalesce:
          rx_frames: 128
          tx_frames: 128
      state: up
```

2. Run the playbook:

```
# ansible-playbook ~/configure-ethernet-device-with-ethtoolcoalesce-settings.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md**

7.14. NETWORK STATES FOR THE NETWORK RHEL SYSTEM ROLE

The **network** RHEL system role supports state configurations in playbooks to configure the devices. For this, use the **network_state** variable followed by the state configurations.

Benefits of using the **network_state** variable in a playbook:

- Using the declarative method with the state configurations, you can configure interfaces, and the NetworkManager creates a profile for these interfaces in the background.
- With the **network_state** variable, you can specify the options that you require to change, and all the other options will remain the same as they are. However, with the **network_connections** variable, you must specify all settings to change the network connection profile.

For example, to create an Ethernet connection with dynamic IP address settings, use the following **vars** block in your playbook:

Playbook with state configurations	Regular playbook

```
vars:
  network_state:
    interfaces:
    - name: enp7s0
      type: ethernet
      state: up
      ipv4:
        enabled: true
        auto-dns: true
        auto-gateway: true
        auto-routes: true
        dhcp: true
      ipv6:
        enabled: true
        auto-dns: true
        auto-gateway: true
        auto-routes: true
        autoconf: true
        dhcp: true
```

```
vars:
  network_connections:
    - name: enp7s0
      interface_name: enp7s0
      type: ethernet
      autoconnect: yes
      ip:
        dhcp4: yes
        auto6: yes
      state: up
```

For example, to only change the connection status of dynamic IP address settings that you created as above, use the following **vars** block in your playbook:

Playbook with state configurations

```
vars:
  network_state:
    interfaces:
    - name: enp7s0
      type: ethernet
      state: down
```

Regular playbook

```
vars:
  network_connections:
    - name: enp7s0
      interface_name: enp7s0
      type: ethernet
      autoconnect: yes
      ip:
        dhcp4: yes
        auto6: yes
      state: down
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md**
- Introduction to Nmstate

CHAPTER 8. USING THE NETWORK RHEL SYSTEM ROLE TO MANAGE INFINIBAND CONNECTIONS

Use the **network** RHEL System Role to configure InfiniBand connections on Red Hat Enterprise Linux.

8.1. CONFIGURING AN IPOIB CONNECTION USING THE NETWORK RHEL SYSTEM ROLE

You can use the **network** RHEL System Role to remotely create NetworkManager connection profiles for IP over InfiniBand (IPoIB) devices.

Prerequisites

- You have prepared the control node and the managed nodes .
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.
- The hosts or host groups on which you want to run this playbook are listed in the Ansible inventory file.
- An InfiniBand device named **mlx4_ib0** is installed in the managed nodes.
- The managed nodes use NetworkManager to configure the network.

Procedure

1. Create a playbook file, for example ***~/IPoIB.yml***, with the following content:

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
  - name: Configure IPoIB
    include_role:
      name: rhel-system-roles.network

    vars:
      network_connections:

        # InfiniBand connection mlx4_ib0
        - name: mlx4_ib0
          interface_name: mlx4_ib0
          type: infiniband

        # IPoIB device mlx4_ib0.8002 on top of mlx4_ib0
        - name: mlx4_ib0.8002
          type: infiniband
          autoconnect: yes
          infiniband:
            p_key: 0x8002
            transport_mode: datagram
          parent: mlx4_ib0
```

```
    ip:
      address:
        - 192.0.2.1/24
        - 2001:db8:1::1/64
    state: up
```

If you set a **p_key** parameter as in this example, do not set an **interface_name** parameter on the IPoIB device.

2. Run the playbook:

```
# ansible-playbook ~/IPoIB.yml
```

Verification

1. On the **managed-node-01.example.com** host, display the IP settings of the **mlx4_ib0.8002** device:

```
# ip address show mlx4_ib0.8002
...
inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute ib0.8002
   valid_lft forever preferred_lft forever
inet6 2001:db8:1::1/64 scope link tentative noprefixroute
   valid_lft forever preferred_lft forever
```

2. Display the partition key (P_Key) of the **mlx4_ib0.8002** device:

```
# cat /sys/class/net/mlx4_ib0.8002/pkey
0x8002
```

3. Display the mode of the **mlx4_ib0.8002** device:

```
# cat /sys/class/net/mlx4_ib0.8002/mode
datagram
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md**

CHAPTER 9. CONFIGURING FIREWALLD USING SYSTEM ROLES

You can use the **firewall** System Role to configure settings of the **firewalld** service on multiple clients at once. This solution:

- Provides an interface with efficient input settings.
- Keeps all intended **firewalld** parameters in one place.

After you run the **firewall** role on the control node, the System Role applies the **firewalld** parameters to the managed node immediately and makes them persistent across reboots.

9.1. INTRODUCTION TO THE FIREWALL RHEL SYSTEM ROLE

RHEL System Roles is a set of contents for the Ansible automation utility. This content together with the Ansible automation utility provides a consistent configuration interface to remotely manage multiple systems.

The **rhel-system-roles.firewall** role from the RHEL System Roles was introduced for automated configurations of the **firewalld** service. The **rhel-system-roles** package contains this System Role, and also the reference documentation.

To apply the **firewalld** parameters on one or more systems in an automated fashion, use the **firewall** System Role variable in a playbook. A playbook is a list of one or more plays that is written in the text-based YAML format.

You can use an inventory file to define a set of systems that you want Ansible to configure.

With the **firewall** role you can configure many different **firewalld** parameters, for example:

- Zones.
- The services for which packets should be allowed.
- Granting, rejection, or dropping of traffic access to ports.
- Forwarding of ports or port ranges for a zone.

Additional resources

- **README.md** and **README.html** files in the **/usr/share/doc/rhel-system-roles/firewall/** directory
- Working with playbooks
- How to build your inventory

9.2. RESETTING THE FIREWALLD SETTINGS USING THE FIREWALL RHEL SYSTEM ROLE

With the **firewall** RHEL system role, you can reset the **firewalld** settings to their default state. If you add the **previous:replaced** parameter to the variable list, the System Role removes all existing user-defined settings and resets **firewalld** to the defaults. If you combine the **previous:replaced** parameter with other settings, the **firewall** role removes all existing settings before applying new ones.

Run this procedure on Ansible control node.

Prerequisites

- The **ansible** and **rhel-system-roles** packages are installed on the control node.
- If you use a different remote user than root when you run the playbook, you must have appropriate sudo permissions on the managed node.
- One or more managed nodes that you configure with the **firewall** RHEL System Role.

Procedure

1. If the host on which you want to execute the instructions in the playbook is not yet inventoried, add the IP or name of this host to the **/etc/ansible/hosts** Ansible inventory file:

```
node.example.com
```

2. Create the ~/**reset-firewalld.yml** playbook with the following content:

```
- name: Reset firewalld example
  hosts: node.example.com
  tasks:
    - name: Reset firewalld
      include_role:
        name: rhel-system-roles.firewall
      vars:
        firewall:
          - previous: replaced
```

3. Run the playbook:

   a. To connect as root user to the managed node:

```
# ansible-playbook -u root ~/reset-firewalld.yml
```

   b. To connect as a user to the managed node:

```
# ansible-playbook -u user_name --ask-become-pass ~/reset-firewalld.yml
```

   The **--ask-become-pass** option makes sure that the **ansible-playbook** command prompts for the sudo password of the user defined in the **-u *user_name*** option.

If you do not specify the **-u *user_name*** option, **ansible-playbook** connects to the managed node as the user that is currently logged in to the control node.

Verification

- Run this command as **root** on the managed node to check all the zones:

```
# firewall-cmd --list-all-zones
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.firewall/README.md**
- **ansible-playbook(1)**
- **firewalld(1)**

9.3. FORWARDING INCOMING TRAFFIC FROM ONE LOCAL PORT TO A DIFFERENT LOCAL PORT

With the **firewall** role you can remotely configure **firewalld** parameters with persisting effect on multiple managed hosts.

Perform this procedure on the Ansible control node.

Prerequisites

- You have prepared the control node and the managed nodes .
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on the them.
- The hosts or host groups on which you want run this playbook are listed in the Ansible inventory file.

Procedure

1. Create a playbook file, for example ***~/port_forwarding.yml***, with the following content:

```
---
- name: Configure firewalld
  hosts: managed-node-01.example.com
  tasks:
  - name: Forward incoming traffic on port 8080 to 443
    include_role:
      name: rhel-system-roles.firewall

    vars:
      firewall:
        - { forward_port: 8080/tcp;443;, state: enabled, runtime: true, permanent: true }
```

2. Run the playbook:

```
# ansible-playbook ~/port_forwarding.yml
```

Verification

- On the managed host, display the **firewalld** settings:

```
# firewall-cmd --list-forward-ports
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.firewall/README.md**

9.4. CONFIGURING PORTS USING SYSTEM ROLES

You can use the RHEL **firewall** System Role to open or close ports in the local firewall for incoming traffic and make the new configuration persist across reboots. The example describes how to configure the default zone to permit incoming traffic for the HTTPS service.

Perform this procedure on the Ansible control node.

Prerequisites

- You have prepared the control node and the managed nodes .
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on the them.
- The hosts or host groups on which you want run this playbook are listed in the Ansible inventory file.

Procedure

1. Create a playbook file, for example *~/**opening-a-port.yml***, with the following content:

```
---
- name: Configure firewalld
  hosts: managed-node-01.example.com
  tasks:
  - name: Allow incoming HTTPS traffic to the local host
    include_role:
      name: rhel-system-roles.firewall

    vars:
      firewall:
        - port: 443/tcp
          service: http
          state: enabled
          runtime: true
          permanent: true
```

   The **permanent: true** option makes the new settings persistent across reboots.

2. Run the playbook:

```
# ansible-playbook ~/opening-a-port.yml
```

Verification

- On the managed node, verify that the **443/tcp** port associated with the **HTTPS** service is open:

```
# firewall-cmd --list-ports
443/tcp
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.firewall/README.md**

9.5. CONFIGURING A DMZ FIREWALLD ZONE BY USING THE FIREWALLD RHEL SYSTEM ROLE

As a system administrator, you can use the **firewall** System Role to configure a **dmz** zone on the **enp1s0** interface to permit **HTTPS** traffic to the zone. In this way, you enable external users to access your web servers.

Perform this procedure on the Ansible control node.

Prerequisites

- You have prepared the control node and the managed nodes .
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on the them.
- The hosts or host groups on which you want run this playbook are listed in the Ansible inventory file.

Procedure

1. Create a playbook file, for example ***~/configuring-a-dmz.yml***, with the following content:

```
---
- name: Configure firewalld
  hosts: managed-node-01.example.com
  tasks:
  - name: Creating a DMZ with access to HTTPS port and masquerading for hosts in DMZ
    include_role:
      name: rhel-system-roles.firewall

    vars:
      firewall:
        - zone: dmz
          interface: enp1s0
          service: https
          state: enabled
          runtime: true
          permanent: true
```

2. Run the playbook:

```
# ansible-playbook ~/configuring-a-dmz.yml
```

Verification

- On the managed node, view detailed information about the **dmz** zone:

```
# firewall-cmd --zone=dmz --list-all
```

```
dmz (active)
  target: default
  icmp-block-inversion: no
  interfaces: enp1s0
  sources:
  services: https ssh
  ports:
  protocols:
  forward: no
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.firewall/README.md**

CHAPTER 10. VARIABLES OF THE POSTFIX ROLE IN SYSTEM ROLES

The **postfix** role variables allow the user to install, configure, and start the **postfix** Mail Transfer Agent (MTA).

The following role variables are defined in this section:

- **postfix_conf**: It includes key/value pairs of all the supported **postfix** configuration parameters. By default, the **postfix_conf** does not have a value.

```
postfix_conf:
  relayhost: example.com
```

If your scenario requires removing any existing configuration and apply the desired configuration on top of a clean **postfix** installation, specify the **previous: replaced** option within the **postfix_conf** dictionary:

An example with the **previous: replaced** option:

```
postfix_conf:
  relayhost: example.com
  previous: replaced
```

- **postfix_check**: It determines if a check has been executed before starting the **postfix** to verify the configuration changes. The default value is true.

For example:

```
postfix_check: true
```

- **postfix_backup**: It determines whether a single backup copy of the configuration is created. By default the **postfix_backup** value is false.

To overwrite any previous backup run the following command:

```
# *cp /etc/postfix/main.cf /etc/postfix/main.cf.backup*
```

If the **postfix_backup** value is changed to **true**, you must also set the **postfix_backup_multiple** value to false.

For example:

```
postfix_backup: true
postfix_backup_multiple: false
```

- **postfix_backup_multiple**: It determines if the role will make a timestamped backup copy of the configuration.

To keep multiple backup copies, run the following command:

```
# *cp /etc/postfix/main.cf /etc/postfix/main.cf.$(date -Isec)*
```

By default the value of **postfix_backup_multiple** is true. The **postfix_backup_multiple:true** setting overrides **postfix_backup**. If you want to use **postfix_backup** you must set the **postfix_backup_multiple:false**.

IMPORTANT

The configuration parameters cannot be removed. Before running the **postfix** role, set the **postfix_conf** to all the required configuration parameters and use the file module to remove **/etc/postfix/main.cf**

10.1. ADDITIONAL RESOURCES

- **/usr/share/doc/rhel-system-roles/postfix/README.md**

CHAPTER 11. CONFIGURING SELINUX USING SYSTEM ROLES

11.1. INTRODUCTION TO THE SELINUX SYSTEM ROLE

RHEL System Roles is a collection of Ansible roles and modules that provide a consistent configuration interface to remotely manage multiple RHEL systems. The **selinux** System Role enables the following actions:

- Cleaning local policy modifications related to SELinux booleans, file contexts, ports, and logins.
- Setting SELinux policy booleans, file contexts, ports, and logins.
- Restoring file contexts on specified files or directories.
- Managing SELinux modules.

The following table provides an overview of input variables available in the **selinux** System Role.

Table 11.1. selinux System Role variables

Role variable	Description	CLI alternative
selinux_policy	Chooses a policy protecting targeted processes or Multi Level Security protection.	**SELINUXTYPE** in **/etc/selinux/config**
selinux_state	Switches SELinux modes.	**setenforce** and **SELINUX** in **/etc/selinux/config**.
selinux_booleans	Enables and disables SELinux booleans.	**setsebool**
selinux_fcontexts	Adds or removes a SELinux file context mapping.	**semanage fcontext**
selinux_restore_dirs	Restores SELinux labels in the file-system tree.	**restorecon -R**
selinux_ports	Sets SELinux labels on ports.	**semanage port**
selinux_logins	Sets users to SELinux user mapping.	**semanage login**
selinux_modules	Installs, enables, disables, or removes SELinux modules.	**semodule**

The **/usr/share/doc/rhel-system-roles/selinux/example-selinux-playbook.yml** example playbook installed by the **rhel-system-roles** package demonstrates how to set the targeted policy in enforcing mode. The playbook also applies several local policy modifications and restores file contexts in the **/tmp/test_dir/** directory.

For a detailed reference on **selinux** role variables, install the **rhel-system-roles** package, and see the **README.md** or **README.html** files in the **/usr/share/doc/rhel-system-roles/selinux/** directory.

Additional resources

- Introduction to RHEL System Roles .

11.2. USING THE SELINUX SYSTEM ROLE TO APPLY SELINUX SETTINGS ON MULTIPLE SYSTEMS

Follow the steps to prepare and apply an Ansible playbook with your verified SELinux settings.

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **selinux** System Role.
- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.
  On the control node:
  - The **ansible-core** and **rhel-system-roles** packages are installed.
  - An inventory file which lists the managed nodes.

IMPORTANT

RHEL 8.0-8.5 provided access to a separate Ansible repository that contains Ansible Engine 2.9 for automation based on Ansible. Ansible Engine contains command-line utilities such as **ansible**, **ansible-playbook**, connectors such as **docker** and **podman**, and many plugins and modules. For information on how to obtain and install Ansible Engine, see the How to download and install Red Hat Ansible Engine Knowledgebase article.

RHEL 8.6 and 9.0 have introduced Ansible Core (provided as the **ansible-core** package), which contains the Ansible command-line utilities, commands, and a small set of built-in Ansible plugins. RHEL provides this package through the AppStream repository, and it has a limited scope of support. For more information, see the Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories Knowledgebase article.

- An inventory file which lists the managed nodes.

Procedure

1. Prepare your playbook. You can either start from the scratch or modify the example playbook installed as a part of the **rhel-system-roles** package:

```
# cp /usr/share/doc/rhel-system-roles/selinux/example-selinux-playbook.yml my-selinux-playbook.yml
# vi my-selinux-playbook.yml
```

2. Change the content of the playbook to fit your scenario. For example, the following part ensures that the system installs and enables the **selinux-local-1.pp** SELinux module:

```
selinux_modules:
- { path: "selinux-local-1.pp", priority: "400" }
```

3. Save the changes, and exit the text editor.

4. Run your playbook on the *host1*, *host2*, and *host3* systems:

```
# ansible-playbook -i host1,host2,host3 my-selinux-playbook.yml
```

Additional resources

- For more information, install the **rhel-system-roles** package, and see the **/usr/share/doc/rhel-system-roles/selinux/** and **/usr/share/ansible/roles/rhel-system-roles.selinux/** directories.

CHAPTER 12. USING THE LOGGING SYSTEM ROLE

As a system administrator, you can use the **logging** System Role to configure a RHEL host as a logging server to collect logs from many client systems.

12.1. THE LOGGING SYSTEM ROLE

With the **logging** System Role, you can deploy logging configurations on local and remote hosts.

To apply a **logging** System Role on one or more systems, you define the logging configuration in a *playbook*. A playbook is a list of one or more plays. Playbooks are human-readable, and they are written in the YAML format. For more information about playbooks, see Working with playbooks in Ansible documentation.

The set of systems that you want to configure according to the playbook is defined in an *inventory file*. For more information on creating and using inventories, see How to build your inventory in Ansible documentation.

Logging solutions provide multiple ways of reading logs and multiple logging outputs.

For example, a logging system can receive the following inputs:

- local files,
- **systemd/journal**,
- another logging system over the network.

In addition, a logging system can have the following outputs:

- logs stored in the local files in the **/var/log** directory,
- logs sent to Elasticsearch,
- logs forwarded to another logging system.

With the **logging** System Role, you can combine the inputs and outputs to fit your scenario. For example, you can configure a logging solution that stores inputs from **journal** in a local file, whereas inputs read from files are both forwarded to another logging system and stored in the local log files.

12.2. LOGGING SYSTEM ROLE PARAMETERS

In a **logging** System Role playbook, you define the inputs in the **logging_inputs** parameter, outputs in the **logging_outputs** parameter, and the relationships between the inputs and outputs in the **logging_flows** parameter. The **logging** System Role processes these variables with additional options to configure the logging system. You can also enable encryption.

NOTE

Currently, the only available logging system in the **logging** System Role is **Rsyslog**.

- **logging_inputs**: List of inputs for the logging solution.

- **name**: Unique name of the input. Used in the **logging_flows**: inputs list and a part of the generated **config** file name.
- **type**: Type of the input element. The type specifies a task type which corresponds to a directory name in **roles/rsyslog/{tasks,vars}/inputs/**.
  - **basics**: Inputs configuring inputs from **systemd** journal or **unix** socket.
    - **kernel_message**: Load **imklog** if set to **true**. Default to **false**.
    - **use_imuxsock**: Use **imuxsock** instead of **imjournal**. Default to **false**.
    - **ratelimit_burst**: Maximum number of messages that can be emitted within **ratelimit_interval**. Default to **20000** if **use_imuxsock** is false. Default to **200** if **use_imuxsock** is true.
    - **ratelimit_interval**: Interval to evaluate **ratelimit_burst**. Default to 600 seconds if **use_imuxsock** is false. Default to 0 if **use_imuxsock** is true. 0 indicates rate limiting is turned off.
    - **persist_state_interval**: Journal state is persisted every **value** messages. Default to **10**. Effective only when **use_imuxsock** is false.
  - **files**: Inputs configuring inputs from local files.
  - **remote**: Inputs configuring inputs from the other logging system over network.
- **state**: State of the configuration file. **present** or **absent**. Default to **present**.

- **logging_outputs**: List of outputs for the logging solution.
  - **files**: Outputs configuring outputs to local files.
  - **forwards**: Outputs configuring outputs to another logging system.
  - **remote_files**: Outputs configuring outputs from another logging system to local files.
- **logging_flows**: List of flows that define relationships between **logging_inputs** and **logging_outputs**. The **logging_flows** variable has the following keys:
  - **name**: Unique name of the flow
  - **inputs**: List of **logging_inputs** name values
  - **outputs**: List of **logging_outputs** name values.

Additional resources

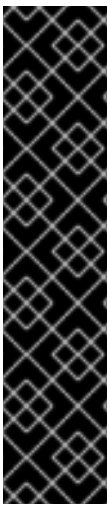
- Documentation installed with the **rhel-system-roles** package in **/usr/share/ansible/roles/rhel-system-roles.logging/README.html**

12.3. APPLYING A LOCAL LOGGING SYSTEM ROLE

Follow these steps to prepare and apply an Ansible playbook to configure a logging solution on a set of separate machines. Each machine will record logs locally.

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **logging** System Role.
- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.
  On the control node:
  - The **ansible-core** and **rhel-system-roles** packages are installed.

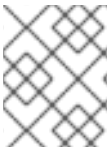


IMPORTANT

RHEL 8.0-8.5 provided access to a separate Ansible repository that contains Ansible Engine 2.9 for automation based on Ansible. Ansible Engine contains command-line utilities such as **ansible**, **ansible-playbook**, connectors such as **docker** and **podman**, and many plugins and modules. For information on how to obtain and install Ansible Engine, see the How to download and install Red Hat Ansible Engine Knowledgebase article.

RHEL 8.6 and 9.0 have introduced Ansible Core (provided as the **ansible-core** package), which contains the Ansible command-line utilities, commands, and a small set of built-in Ansible plugins. RHEL provides this package through the AppStream repository, and it has a limited scope of support. For more information, see the Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories Knowledgebase article.

- An inventory file which lists the managed nodes.



NOTE

You do not have to have the **rsyslog** package installed, because the System Role installs **rsyslog** when deployed.

Procedure

1. Create a playbook that defines the required role:
   a. Create a new YAML file and open it in a text editor, for example:

```
# vi logging-playbook.yml
```

   b. Insert the following content:

```
---
- name: Deploying basics input and implicit files output
  hosts: all
  roles:
    - rhel-system-roles.logging
  vars:
    logging_inputs:
      - name: system_input
        type: basics
    logging_outputs:
      - name: files_output
        type: files
    logging_flows:
```

```
    - name: flow1
      inputs: [system_input]
      outputs: [files_output]
```

2. Run the playbook on a specific inventory:

```
# ansible-playbook -i inventory-file /path/to/file/logging-playbook.yml
```

Where:

- ***inventory-file*** is the inventory file.
- ***logging-playbook.yml*** is the playbook you use.

Verification

1. Test the syntax of the **/etc/rsyslog.conf** file:

```
# rsyslogd -N 1
rsyslogd: version 8.1911.0-6.el8, config validation run (level 1), master config
/etc/rsyslog.conf
rsyslogd: End of config validation run. Bye.
```

2. Verify that the system sends messages to the log:

   a. Send a test message:

```
# logger test
```

   b. View the **/var/log/messages** log, for example:

```
# cat /var/log/messages
Aug  5 13:48:31 hostname root[6778]: test
```

   Where `*hostname*` is the host name of the client system. Note that the log contains the user name of the user that entered the logger command, in this case **root**.

12.4. FILTERING LOGS IN A LOCAL LOGGING SYSTEM ROLE

You can deploy a logging solution which filters the logs based on the **rsyslog** property-based filter.

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **logging** System Role.
- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.
  On the control node:
  - Red Hat Ansible Core is installed
  - The **rhel-system-roles** package is installed

- An inventory file which lists the managed nodes.

NOTE

You do not have to have the **rsyslog** package installed, because the System Role installs **rsyslog** when deployed.

Procedure

1. Create a new ***playbook.yml*** file with the following content:

```
---
- name: Deploying files input and configured files output
  hosts: all
  roles:
    - linux-system-roles.logging
  vars:
    logging_inputs:
      - name: files_input
        type: basics
    logging_outputs:
      - name: files_output0
        type: files
        property: msg
        property_op: contains
        property_value: error
        path: /var/log/errors.log
      - name: files_output1
        type: files
        property: msg
        property_op: "!contains"
        property_value: error
        path: /var/log/others.log
    logging_flows:
      - name: flow0
        inputs: [files_input]
        outputs: [files_output0, files_output1]
```

Using this configuration, all messages that contain the ***error*** string are logged in **/var/log/errors.log**, and all other messages are logged in **/var/log/others.log**.

You can replace the ***error*** property value with the string by which you want to filter.

You can modify the variables according to your preferences.

2. Optional: Verify playbook syntax.

```
# ansible-playbook --syntax-check playbook.yml
```

3. Run the playbook on your inventory file:

```
# ansible-playbook -i inventory_file /path/to/file/playbook.yml
```

Verification

1. Test the syntax of the **/etc/rsyslog.conf** file:

```
# rsyslogd -N 1
rsyslogd: version 8.1911.0-6.el8, config validation run (level 1), master config
/etc/rsyslog.conf
rsyslogd: End of config validation run. Bye.
```

2. Verify that the system sends messages that contain the **error** string to the log:

   a. Send a test message:

   ```
   # logger error
   ```

   b. View the **/var/log/errors.log** log, for example:

   ```
   # cat /var/log/errors.log
   Aug  5 13:48:31 hostname root[6778]: error
   ```

   Where ***hostname*** is the host name of the client system. Note that the log contains the user name of the user that entered the logger command, in this case **root**.

Additional resources

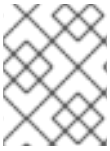
- Documentation installed with the **rhel-system-roles** package in **/usr/share/ansible/roles/rhel-system-roles.logging/README.html**

12.5. APPLYING A REMOTE LOGGING SOLUTION USING THE LOGGING SYSTEM ROLE

Follow these steps to prepare and apply a Red Hat Ansible Core playbook to configure a remote logging solution. In this playbook, one or more clients take logs from **systemd-journal** and forward them to a remote server. The server receives remote input from **remote_rsyslog** and **remote_files** and outputs the logs to local files in directories named by remote host names.

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **logging** System Role.
- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.
  On the control node:
  - The **ansible-core** and **rhel-system-roles** packages are installed.
  - An inventory file which lists the managed nodes.



NOTE

You do not have to have the **rsyslog** package installed, because the System Role installs **rsyslog** when deployed.

Procedure

1. Create a playbook that defines the required role:

   a. Create a new YAML file and open it in a text editor, for example:

   ```
   # vi logging-playbook.yml
   ```

   b. Insert the following content into the file:

   ```
   ---
   - name: Deploying remote input and remote_files output
     hosts: server
     roles:
       - rhel-system-roles.logging
     vars:
       logging_inputs:
         - name: remote_udp_input
           type: remote
           udp_ports: [ 601 ]
         - name: remote_tcp_input
           type: remote
           tcp_ports: [ 601 ]
       logging_outputs:
         - name: remote_files_output
           type: remote_files
       logging_flows:
         - name: flow_0
           inputs: [remote_udp_input, remote_tcp_input]
           outputs: [remote_files_output]

   - name: Deploying basics input and forwards output
     hosts: clients
     roles:
       - rhel-system-roles.logging
     vars:
       logging_inputs:
         - name: basic_input
           type: basics
       logging_outputs:
         - name: forward_output0
           type: forwards
           severity: info
           target: _host1.example.com_
           udp_port: 601
         - name: forward_output1
           type: forwards
           facility: mail
           target: _host1.example.com_
           tcp_port: 601
       logging_flows:
         - name: flows0
           inputs: [basic_input]
           outputs: [forward_output0, forward_output1]

   [basic_input]
   [forward_output0, forward_output1]
   ```

Where ***host1.example.com*** is the logging server.

NOTE

You can modify the parameters in the playbook to fit your needs.

WARNING

The logging solution works only with the ports defined in the SELinux policy of the server or client system and open in the firewall. The default SELinux policy includes ports 601, 514, 6514, 10514, and 20514. To use a different port, modify the SELinux policy on the client and server systems. Configuring the firewall through System Roles is not yet supported.

2. Create an inventory file that lists your servers and clients:

   a. Create a new file and open it in a text editor, for example:

   ```
   # vi inventory.ini
   ```

   b. Insert the following content into the inventory file:

   ```
   [servers]
   server ansible_host=host1.example.com
   [clients]
   client ansible_host=host2.example.com
   ```

   Where:

   - ***host1.example.com*** is the logging server.
   - ***host2.example.com*** is the logging client.

3. Run the playbook on your inventory.

   ```
   # ansible-playbook -i /path/to/file/inventory.ini /path/to/file/_logging-playbook.yml
   ```

   Where:

   - ***inventory.ini*** is the inventory file.
   - ***logging-playbook.yml*** is the playbook you created.

Verification

1. On both the client and the server system, test the syntax of the **/etc/rsyslog.conf** file:

   ```
   # rsyslogd -N 1
   rsyslogd: version 8.1911.0-6.el8, config validation run (level 1), master config
   ```

```
/etc/rsyslog.conf
rsyslogd: End of config validation run. Bye.
```

2. Verify that the client system sends messages to the server:

   a. On the client system, send a test message:

   ```
   # logger test
   ```

   b. On the server system, view the **/var/log/messages** log, for example:

   ```
   # cat /var/log/messages
   Aug  5 13:48:31 host2.example.com root[6778]: test
   ```

   Where ***host2.example.com*** is the host name of the client system. Note that the log contains the user name of the user that entered the logger command, in this case **root**.

Additional resources

- Preparing a control node and managed nodes to use RHEL System Roles
- Documentation installed with the **rhel-system-roles** package in **/usr/share/ansible/roles/rhel-system-roles.logging/README.html**
- RHEL System Roles KB article

12.6. USING THE LOGGING SYSTEM ROLE WITH TLS

Transport Layer Security (TLS) is a cryptographic protocol designed to securely communicate over the computer network.

As an administrator, you can use the **logging** RHEL System Role to configure secure transfer of logs using Red Hat Ansible Automation Platform.

12.6.1. Configuring client logging with TLS

You can use the **logging** System Role to configure logging in RHEL systems that are logged on a local machine and can transfer logs to the remote logging system with TLS by running an Ansible playbook.

This procedure configures TLS on all hosts in the clients group in the Ansible inventory. The TLS protocol encrypts the message transmission for secure transfer of logs over the network.

Prerequisites

- You have permissions to run playbooks on managed nodes on which you want to configure TLS.
- The managed nodes are listed in the inventory file on the control node.
- The **ansible** and **rhel-system-roles** packages are installed on the control node.

Procedure

1. Create a ***playbook.yml*** file with the following content:

```
---
- name: Deploying files input and forwards output with certs
  hosts: clients
  roles:
    - rhel-system-roles.logging
  vars:
    logging_pki_files:
      - ca_cert_src: /local/path/to/ca_cert.pem
        cert_src: /local/path/to/cert.pem
        private_key_src: /local/path/to/key.pem
    logging_inputs:
      - name: input_name
        type: files
        input_log_path: /var/log/containers/*.log
    logging_outputs:
      - name: output_name
        type: forwards
        target: your_target_host
        tcp_port: 514
        tls: true
        pki_authmode: x509/name
        permitted_server: 'server.example.com'
    logging_flows:
      - name: flow_name
        inputs: [input_name]
        outputs: [output_name]
```

The playbook uses the following parameters:

logging_pki_files

Using this parameter you can configure TLS and has to pass **ca_cert_src**, **cert_src**, and **private_key_src** parameters.

ca_cert

Represents the path to CA certificate. Default path is **/etc/pki/tls/certs/ca.pem** and the file name is set by the user.

cert

Represents the path to cert. Default path is **/etc/pki/tls/certs/server-cert.pem** and the file name is set by the user.

private_key

Represents the path to private key. Default path is **/etc/pki/tls/private/server-key.pem** and the file name is set by the user.

ca_cert_src

Represents local CA cert file path which is copied to the target host. If **ca_cert** is specified, it is copied to the location.

cert_src

Represents the local cert file path which is copied to the target host. If **cert** is specified, it is copied to the location.

private_key_src

Represents the local key file path which is copied to the target host. If **private_key** is specified, it is copied to the location.

tls

Using this parameter ensures secure transfer of logs over the network. If you do not want a secure wrapper, you can set **tls: true**.

2. Verify playbook syntax:

```
# ansible-playbook --syntax-check playbook.yml
```

3. Run the playbook on your inventory file:

```
# ansible-playbook -i inventory_file playbook.yml
```

12.6.2. Configuring server logging with TLS

You can use the **logging** System Role to configure logging in RHEL systems as a server and can receive logs from the remote logging system with TLS by running an Ansible playbook.

This procedure configures TLS on all hosts in the server group in the Ansible inventory.

Prerequisites

- You have permissions to run playbooks on managed nodes on which you want to configure TLS.
- The managed nodes are listed in the inventory file on the control node.
- The **ansible** and **rhel-system-roles** packages are installed on the control node.

Procedure

1. Create a ***playbook.yml*** file with the following content:

```
---
- name: Deploying remote input and remote_files output with certs
  hosts: server
  roles:
    - rhel-system-roles.logging
  vars:
    logging_pki_files:
      - ca_cert_src: /local/path/to/ca_cert.pem
        cert_src: /local/path/to/cert.pem
        private_key_src: /local/path/to/key.pem
    logging_inputs:
      - name: input_name
        type: remote
        tcp_ports: 514
        tls: true
        permitted_clients: ['clients.example.com']
    logging_outputs:
      - name: output_name
        type: remote_files
        remote_log_path: /var/log/remote/%FROMHOST%/%PROGRAMNAME:::secpath-
replace%.log
        async_writing: true
        client_count: 20
        io_buffer_size: 8192
    logging_flows:
```

```
    - name: flow_name
      inputs: [input_name]
      outputs: [output_name]
```

The playbook uses the following parameters:

logging_pki_files

Using this parameter you can configure TLS and has to pass **ca_cert_src**, **cert_src**, and **private_key_src** parameters.

ca_cert

Represents the path to CA certificate. Default path is **/etc/pki/tls/certs/ca.pem** and the file name is set by the user.

cert

Represents the path to cert. Default path is **/etc/pki/tls/certs/server-cert.pem** and the file name is set by the user.

private_key

Represents the path to private key. Default path is **/etc/pki/tls/private/server-key.pem** and the file name is set by the user.

ca_cert_src

Represents local CA cert file path which is copied to the target host. If **ca_cert** is specified, it is copied to the location.

cert_src

Represents the local cert file path which is copied to the target host. If **cert** is specified, it is copied to the location.

private_key_src

Represents the local key file path which is copied to the target host. If **private_key** is specified, it is copied to the location.

tls

Using this parameter ensures secure transfer of logs over the network. If you do not want a secure wrapper, you can set **tls: true**.

2. Verify playbook syntax:

```
# ansible-playbook --syntax-check playbook.yml
```

3. Run the playbook on your inventory file:

```
# ansible-playbook -i inventory_file playbook.yml
```

12.7. USING THE LOGGING SYSTEM ROLES WITH RELP

Reliable Event Logging Protocol (RELP) is a networking protocol for data and message logging over the TCP network. It ensures reliable delivery of event messages and you can use it in environments that do not tolerate any message loss.

The RELP sender transfers log entries in form of commands and the receiver acknowledges them once they are processed. To ensure consistency, RELP stores the transaction number to each transferred command for any kind of message recovery.

You can consider a remote logging system in between the RELP Client and RELP Server. The RELP Client transfers the logs to the remote logging system and the RELP Server receives all the logs sent by the remote logging system.

Administrators can use the **logging** System Role to configure the logging system to reliably send and receive log entries.

12.7.1. Configuring client logging with RELP

You can use the **logging** System Role to configure logging in RHEL systems that are logged on a local machine and can transfer logs to the remote logging system with RELP by running an Ansible playbook.

This procedure configures RELP on all hosts in the **clients** group in the Ansible inventory. The RELP configuration uses Transport Layer Security (TLS) to encrypt the message transmission for secure transfer of logs over the network.

Prerequisites

- You have permissions to run playbooks on managed nodes on which you want to configure RELP.
- The managed nodes are listed in the inventory file on the control node.
- The **ansible** and **rhel-system-roles** packages are installed on the control node.

Procedure

1. Create a ***playbook.yml*** file with the following content:

```
---
- name: Deploying basic input and relp output
  hosts: clients
  roles:
    - rhel-system-roles.logging
  vars:
    logging_inputs:
      - name: basic_input
        type: basics
    logging_outputs:
      - name: relp_client
        type: relp
        target: _logging.server.com_
        port: 20514
        tls: true
        ca_cert: _/etc/pki/tls/certs/ca.pem_
        cert: _/etc/pki/tls/certs/client-cert.pem_
        private_key: _/etc/pki/tls/private/client-key.pem_
        pki_authmode: name
        permitted_servers:
          - '*.server.example.com'
    logging_flows:
      - name: _example_flow_
        inputs: [basic_input]
        outputs: [relp_client]
```

The playbooks uses following settings:

- **target**: This is a required parameter that specifies the host name where the remote logging system is running.
- **port**: Port number the remote logging system is listening.
- **tls**: Ensures secure transfer of logs over the network. If you do not want a secure wrapper you can set the **tls** variable to **false**. By default **tls** parameter is set to true while working with RELP and requires key/certificates and triplets {**ca_cert**, **cert**, **private_key**} and/or {**ca_cert_src**, **cert_src**, **private_key_src**}.
  - If {**ca_cert_src**, **cert_src**, **private_key_src**} triplet is set, the default locations **/etc/pki/tls/certs** and **/etc/pki/tls/private** are used as the destination on the managed node to transfer files from control node. In this case, the file names are identical to the original ones in the triplet
  - If {**ca_cert**, **cert**, **private_key**} triplet is set, files are expected to be on the default path before the logging configuration.
  - If both the triplets are set, files are transferred from local path from control node to specific path of the managed node.
- **ca_cert**: Represents the path to CA certificate. Default path is **/etc/pki/tls/certs/ca.pem** and the file name is set by the user.
- **cert**: Represents the path to cert. Default path is **/etc/pki/tls/certs/server-cert.pem** and the file name is set by the user.
- **private_key**: Represents the path to private key. Default path is **/etc/pki/tls/private/server-key.pem** and the file name is set by the user.
- **ca_cert_src**: Represents local CA cert file path which is copied to the target host. If ca_cert is specified, it is copied to the location.
- **cert_src**: Represents the local cert file path which is copied to the target host. If cert is specified, it is copied to the location.
- **private_key_src**: Represents the local key file path which is copied to the target host. If private_key is specified, it is copied to the location.
- **pki_authmode**: Accepts the authentication mode as **name** or **fingerprint**.
- **permitted_servers**: List of servers that will be allowed by the logging client to connect and send logs over TLS.
- **inputs**: List of logging input dictionary.
- **outputs**: List of logging output dictionary.

2. Optional: Verify playbook syntax.

```
# ansible-playbook --syntax-check playbook.yml
```

3. Run the playbook:

```
# ansible-playbook -i inventory_file playbook.yml
```

12.7.2. Configuring server logging with RELP

You can use the **logging** System Role to configure logging in RHEL systems as a server and can receive logs from the remote logging system with RELP by running an Ansible playbook.

This procedure configures RELP on all hosts in the **server** group in the Ansible inventory. The RELP configuration uses TLS to encrypt the message transmission for secure transfer of logs over the network.

Prerequisites

- You have permissions to run playbooks on managed nodes on which you want to configure RELP.
- The managed nodes are listed in the inventory file on the control node.
- The **ansible** and **rhel-system-roles** packages are installed on the control node.

Procedure

1. Create a ***playbook.yml*** file with the following content:

```
---
- name: Deploying remote input and remote_files output
  hosts: server
  roles:
    - rhel-system-roles.logging
  vars:
    logging_inputs:
      - name: relp_server
        type: relp
        port: 20514
        tls: true
        ca_cert: _/etc/pki/tls/certs/ca.pem_
        cert: _/etc/pki/tls/certs/server-cert.pem_
        private_key: _/etc/pki/tls/private/server-key.pem_
        pki_authmode: name
        permitted_clients:
          - '_*example.client.com_'
    logging_outputs:
      - name: _remote_files_output_
        type: _remote_files_
    logging_flows:
      - name: _example_flow_
        inputs: _relp_server_
        outputs: _remote_files_output_
```

The playbooks uses following settings:

- **port**: Port number the remote logging system is listening.
- **tls**: Ensures secure transfer of logs over the network. If you do not want a secure wrapper you can set the **tls** variable to **false**. By default **tls** parameter is set to true while working with RELP and requires key/certificates and triplets {**ca_cert**, **cert**, **private_key**} and/or {**ca_cert_src**, **cert_src**, **private_key_src**}.

    - If {**ca_cert_src**, **cert_src**, **private_key_src**} triplet is set, the default locations **/etc/pki/tls/certs** and **/etc/pki/tls/private** are used as the destination on the managed node to transfer files from control node. In this case, the file names are identical to the original ones in the triplet
    - If {**ca_cert**, **cert**, **private_key**} triplet is set, files are expected to be on the default path before the logging configuration.
    - If both the triplets are set, files are transferred from local path from control node to specific path of the managed node.

- **ca_cert**: Represents the path to CA certificate. Default path is **/etc/pki/tls/certs/ca.pem** and the file name is set by the user.
- **cert**: Represents the path to cert. Default path is **/etc/pki/tls/certs/server-cert.pem** and the file name is set by the user.
- **private_key**: Represents the path to private key. Default path is **/etc/pki/tls/private/server-key.pem** and the file name is set by the user.
- **ca_cert_src**: Represents local CA cert file path which is copied to the target host. If ca_cert is specified, it is copied to the location.
- **cert_src**: Represents the local cert file path which is copied to the target host. If cert is specified, it is copied to the location.
- **private_key_src**: Represents the local key file path which is copied to the target host. If private_key is specified, it is copied to the location.
- **pki_authmode**: Accepts the authentication mode as **name** or **fingerprint**.
- **permitted_clients**: List of clients that will be allowed by the logging server to connect and send logs over TLS.
- **inputs**: List of logging input dictionary.
- **outputs**: List of logging output dictionary.

2. Optional: Verify playbook syntax.

```
# ansible-playbook --syntax-check playbook.yml
```

3. Run the playbook:

```
# ansible-playbook -i inventory_file playbook.yml
```

12.8. ADDITIONAL RESOURCES

- Preparing a control node and managed nodes to use RHEL System Roles
- Documentation installed with the **rhel-system-roles** package in **/usr/share/ansible/roles/rhel-system-roles.logging/README.html**.
- RHEL System Roles
- **ansible-playbook(1)** man page.

CHAPTER 13. CONFIGURING SECURE COMMUNICATION WITH THE SSH SYSTEM ROLES

As an administrator, you can use the **sshd** System Role to configure SSH servers and the **ssh** System Role to configure SSH clients consistently on any number of RHEL systems at the same time by using Red Hat Ansible Automation Platform.

13.1. SSH SERVER SYSTEM ROLE VARIABLES

In an **sshd** System Role playbook, you can define the parameters for the SSH configuration file according to your preferences and limitations.

If you do not configure these variables, the System Role produces an **sshd_config** file that matches the RHEL defaults.

In all cases, Booleans correctly render as **yes** and **no** in **sshd** configuration. You can define multi-line configuration items using lists. For example:

```
sshd_ListenAddress:
  - 0.0.0.0
  - '::'
```

renders as:

```
ListenAddress 0.0.0.0
ListenAddress ::
```

Variables for the sshd System Role

sshd_enable

If set to **False**, the role is completely disabled. Defaults to **True**.

sshd_skip_defaults

If set to **True**, the System Role does not apply default values. Instead, you specify the complete set of configuration defaults by using either the **sshd** dict, or **sshd_Key** variables. Defaults to **False**.

sshd_manage_service

If set to **False**, the service is not managed, which means it is not enabled on boot and does not start or reload. Defaults to **True** except when running inside a container or AIX, because the Ansible service module does not currently support **enabled** for AIX.

sshd_allow_reload

If set to **False**, **sshd** does not reload after a change of configuration. This can help with troubleshooting. To apply the changed configuration, reload **sshd** manually. Defaults to the same value as **sshd_manage_service** except on AIX, where **sshd_manage_service** defaults to **False** but **sshd_allow_reload** defaults to **True**.

sshd_install_service

If set to **True**, the role installs service files for the **sshd** service. This overrides files provided in the operating system. Do not set to **True** unless you are configuring a second instance and you also change the **sshd_service** variable. Defaults to **False**.

The role uses the files pointed by the following variables as templates:

```
sshd_service_template_service (default: templates/sshd.service.j2)
sshd_service_template_at_service (default: templates/sshd@.service.j2)
sshd_service_template_socket (default: templates/sshd.socket.j2)
```

sshd_service

This variable changes the **sshd** service name, which is useful for configuring a second **sshd** service instance.

sshd

A dict that contains configuration. For example:

```
sshd:
  Compression: yes
  ListenAddress:
    - 0.0.0.0
```

sshd_*OptionName*

You can define options by using simple variables consisting of the **sshd_** prefix and the option name instead of a dict. The simple variables override values in the **sshd** dict.. For example:

```
sshd_Compression: no
```

sshd_match and **sshd_match_1** to **sshd_match_9**

A list of dicts or just a dict for a Match section. Note that these variables do not override match blocks as defined in the **sshd** dict. All of the sources will be reflected in the resulting configuration file.

Secondary variables for the sshd System Role

You can use these variables to override the defaults that correspond to each supported platform.

sshd_packages

You can override the default list of installed packages using this variable.

sshd_config_owner, sshd_config_group, and **sshd_config_mode**

You can set the ownership and permissions for the **openssh** configuration file that this role produces using these variables.

sshd_config_file

The path where this role saves the **openssh** server configuration produced.

sshd_config_namespace

The default value of this variable is null, which means that the role defines the entire content of the configuration file including system defaults. Alternatively, you can use this variable to invoke this role from other roles or from multiple places in a single playbook on systems that do not support drop-in directory. The **sshd_skip_defaults** variable is ignored and no system defaults are used in this case. When this variable is set, the role places the configuration that you specify to configuration snippets in an existing configuration file under the given namespace. If your scenario requires applying the role several times, you need to select a different namespace for each application.

NOTE

Limitations of the **openssh** configuration file still apply. For example, only the first option specified in a configuration file is effective for most of the configuration options.

Technically, the role places snippets in "Match all" blocks, unless they contain other match blocks, to ensure they are applied regardless of the previous match blocks in the existing configuration file. This allows configuring any non-conflicting options from different roles invocations.

sshd_binary

The path to the **sshd** executable of **openssh**.

sshd_service

The name of the **sshd** service. By default, this variable contains the name of the **sshd** service that the target platform uses. You can also use it to set the name of the custom **sshd** service when the role uses the **sshd_install_service** variable.

sshd_verify_hostkeys

Defaults to **auto**. When set to **auto**, this lists all host keys that are present in the produced configuration file, and generates any paths that are not present. Additionally, permissions and file owners are set to default values. This is useful if the role is used in the deployment stage to make sure the service is able to start on the first attempt. To disable this check, set this variable to an empty list **[]**.

sshd_hostkey_owner, sshd_hostkey_group, sshd_hostkey_mode

Use these variables to set the ownership and permissions for the host keys from **sshd_verify_hostkeys**.

sshd_sysconfig

On RHEL-based systems, this variable configures additional details of the **sshd** service. If set to **true**, this role manages also the **/etc/sysconfig/sshd** configuration file based on the following configuration. Defaults to **false**.

sshd_sysconfig_override_crypto_policy

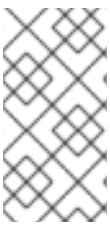
In RHEL, when set to **true**, this variable overrides the system-wide crypto policy. Defaults to **false**.

sshd_sysconfig_use_strong_rng

On RHEL-based systems, this variable can force **sshd** to reseed the **openssl** random number generator with the number of bytes given as the argument. The default is **0**, which disables this functionality. Do not turn this on if the system does not have a hardware random number generator.

13.2. CONFIGURING OPENSSH SERVERS USING THE SSHD SYSTEM ROLE

You can use the **sshd** System Role to configure multiple SSH servers by running an Ansible playbook.



NOTE

You can use the **sshd** System Role with other System Roles that change SSH and SSHD configuration, for example the Identity Management RHEL System Roles. To prevent the configuration from being overwritten, make sure that the **sshd** role uses namespaces (RHEL 8 and earlier versions) or a drop-in directory (RHEL 9).

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **sshd** System Role.
- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.
  On the control node:
  - The **ansible-core** and **rhel-system-roles** packages are installed.

IMPORTANT

RHEL 8.0-8.5 provided access to a separate Ansible repository that contains Ansible Engine 2.9 for automation based on Ansible. Ansible Engine contains command-line utilities such as **ansible**, **ansible-playbook**, connectors such as **docker** and **podman**, and many plugins and modules. For information on how to obtain and install Ansible Engine, see the How to download and install Red Hat Ansible Engine Knowledgebase article.

RHEL 8.6 and 9.0 have introduced Ansible Core (provided as the **ansible-core** package), which contains the Ansible command-line utilities, commands, and a small set of built-in Ansible plugins. RHEL provides this package through the AppStream repository, and it has a limited scope of support. For more information, see the Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories Knowledgebase article.

- An inventory file which lists the managed nodes.

Procedure

1. Copy the example playbook for the **sshd** System Role:

```
# cp /usr/share/doc/rhel-system-roles/sshd/example-root-login-playbook.yml path/custom-playbook.yml
```

2. Open the copied playbook by using a text editor, for example:

```
# vim path/custom-playbook.yml

---
- hosts: all
  tasks:
  - name: Configure sshd to prevent root and password login except from particular subnet
    include_role:
      name: rhel-system-roles.sshd
    vars:
      sshd:
        # root login and password login is enabled only from a particular subnet
        PermitRootLogin: no
        PasswordAuthentication: no
        Match:
        - Condition: "Address 192.0.2.0/24"
          PermitRootLogin: yes
          PasswordAuthentication: yes
```

The playbook configures the managed node as an SSH server configured so that:

- password and **root** user login is disabled
- password and **root** user login is enabled only from the subnet **192.0.2.0/24**

You can modify the variables according to your preferences. For more details, see SSH Server System Role variables .

3. Optional: Verify playbook syntax.

```
# ansible-playbook --syntax-check path/custom-playbook.yml
```

4. Run the playbook on your inventory file:

```
# ansible-playbook -i inventory_file path/custom-playbook.yml

...

PLAY RECAP
**************************************************

localhost : ok=12 changed=2 unreachable=0 failed=0
skipped=10 rescued=0 ignored=0
```

Verification

1. Log in to the SSH server:

```
$ ssh user1@10.1.1.1
```

Where:

- ***user1*** is a user on the SSH server.
- ***10.1.1.1*** is the IP address of the SSH server.

2. Check the contents of the **sshd_config** file on the SSH server:

```
$ vim /etc/ssh/sshd_config

# Ansible managed
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_ecdsa_key
HostKey /etc/ssh/ssh_host_ed25519_key
AcceptEnv LANG LC_CTYPE LC_NUMERIC LC_TIME LC_COLLATE LC_MONETARY
LC_MESSAGES
AcceptEnv LC_PAPER LC_NAME LC_ADDRESS LC_TELEPHONE LC_MEASUREMENT
AcceptEnv LC_IDENTIFICATION LC_ALL LANGUAGE
AcceptEnv XMODIFIERS
AuthorizedKeysFile .ssh/authorized_keys
ChallengeResponseAuthentication no
GSSAPIAuthentication yes
GSSAPICleanupCredentials no
PasswordAuthentication no
PermitRootLogin no
PrintMotd no
```

```
Subsystem sftp /usr/libexec/openssh/sftp-server
SyslogFacility AUTHPRIV
UsePAM yes
X11Forwarding yes
Match Address 192.0.2.0/24
  PasswordAuthentication yes
  PermitRootLogin yes
```

3. Check that you can connect to the server as root from the **192.0.2.0/24** subnet:

   a. Determine your IP address:

   ```
   $ hostname -I
   192.0.2.1
   ```

   If the IP address is within the **192.0.2.1** - **192.0.2.254** range, you can connect to the server.

   b. Connect to the server as **root**:

   ```
   $ ssh root@10.1.1.1
   ```

Additional resources

- **/usr/share/doc/rhel-system-roles/sshd/README.md** file.
- **ansible-playbook(1)** man page.

13.3. SSH SYSTEM ROLE VARIABLES

In an **ssh** System Role playbook, you can define the parameters for the client SSH configuration file according to your preferences and limitations.

If you do not configure these variables, the System Role produces a global **ssh_config** file that matches the RHEL defaults.

In all cases, booleans correctly render as **yes** or **no** in **ssh** configuration. You can define multi-line configuration items using lists. For example:

```
LocalForward:
  - 22 localhost:2222
  - 403 localhost:4003
```

renders as:

```
LocalForward 22 localhost:2222
LocalForward 403 localhost:4003
```

NOTE

The configuration options are case sensitive.

Variables for the ssh System Role

ssh_user

You can define an existing user name for which the System Role modifies user-specific configuration. The user-specific configuration is saved in **~/.ssh/config** of the given user. The default value is null, which modifies global configuration for all users.

ssh_skip_defaults

Defaults to **auto**. If set to **auto**, the System Role writes the system-wide configuration file **/etc/ssh/ssh_config** and keeps the RHEL defaults defined there. Creating a drop-in configuration file, for example by defining the **ssh_drop_in_name** variable, automatically disables the **ssh_skip_defaults** variable.

ssh_drop_in_name

Defines the name for the drop-in configuration file, which is placed in the system-wide drop-in directory. The name is used in the template **/etc/ssh/ssh_config.d/{ssh_drop_in_name}.conf** to reference the configuration file to be modified. If the system does not support drop-in directory, the default value is null. If the system supports drop-in directories, the default value is **00-ansible**.

WARNING

If the system does not support drop-in directories, setting this option will make the play fail.

The suggested format is **NN-name**, where **NN** is a two-digit number used for ordering the configuration files and **name** is any descriptive name for the content or the owner of the file.

ssh

A dict that contains configuration options and their respective values.

ssh_*OptionName*

You can define options by using simple variables consisting of the **ssh_** prefix and the option name instead of a dict. The simple variables override values in the **ssh** dict.

ssh_additional_packages

This role automatically installs the **openssh** and **openssh-clients** packages, which are needed for the most common use cases. If you need to install additional packages, for example, **openssh-keysign** for host-based authentication, you can specify them in this variable.

ssh_config_file

The path to which the role saves the configuration file produced. Default value:

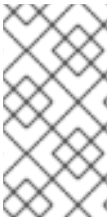
- If the system has a drop-in directory, the default value is defined by the template **/etc/ssh/ssh_config.d/{ssh_drop_in_name}.conf**.
- If the system does not have a drop-in directory, the default value is **/etc/ssh/ssh_config**.
- if the **ssh_user** variable is defined, the default value is **~/.ssh/config**.

ssh_config_owner, ssh_config_group, ssh_config_mode

The owner, group and modes of the created configuration file. By default, the owner of the file is **root:root**, and the mode is **0644**. If **ssh_user** is defined, the mode is **0600**, and the owner and group are derived from the user name specified in the **ssh_user** variable.

13.4. CONFIGURING OPENSSH CLIENTS USING THE SSH SYSTEM ROLE

You can use the **ssh** System Role to configure multiple SSH clients by running an Ansible playbook.

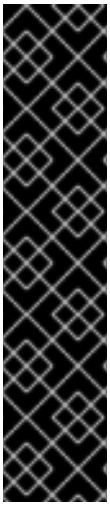


NOTE

You can use the **ssh** System Role with other System Roles that change SSH and SSHD configuration, for example the Identity Management RHEL System Roles. To prevent the configuration from being overwritten, make sure that the **ssh** role uses a drop-in directory (default from RHEL 8).

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **ssh** System Role.
- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.
  On the control node:
  - The **ansible-core** and **rhel-system-roles** packages are installed.



IMPORTANT

RHEL 8.0-8.5 provided access to a separate Ansible repository that contains Ansible Engine 2.9 for automation based on Ansible. Ansible Engine contains command-line utilities such as **ansible**, **ansible-playbook**, connectors such as **docker** and **podman**, and many plugins and modules. For information on how to obtain and install Ansible Engine, see the How to download and install Red Hat Ansible Engine Knowledgebase article.

RHEL 8.6 and 9.0 have introduced Ansible Core (provided as the **ansible-core** package), which contains the Ansible command-line utilities, commands, and a small set of built-in Ansible plugins. RHEL provides this package through the AppStream repository, and it has a limited scope of support. For more information, see the Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories Knowledgebase article.

- An inventory file which lists the managed nodes.

Procedure

1. Create a new ***playbook.yml*** file with the following content:

```
---
- hosts: all
  tasks:
  - name: "Configure ssh clients"
    include_role:
      name: rhel-system-roles.ssh
    vars:
      ssh_user: root
      ssh:
        Compression: true
        GSSAPIAuthentication: no
```

```
    ControlMaster: auto
    ControlPath: ~/.ssh/.cm%C
    Host:
      - Condition: example
        Hostname: example.com
        User: user1
  ssh_ForwardX11: no
```

This playbook configures the **root** user's SSH client preferences on the managed nodes with the following configurations:

- Compression is enabled.
- ControlMaster multiplexing is set to **auto**.
- The ***example*** alias for connecting to the ***example.com*** host is ***user1***.
- The ***example*** host alias is created, which represents a connection to the ***example.com*** host the with ***user1*** user name.
- X11 forwarding is disabled.

Optionally, you can modify these variables according to your preferences. For more details, see **ssh** System Role variables .

2. Optional: Verify playbook syntax.

```
# ansible-playbook --syntax-check path/custom-playbook.yml
```

3. Run the playbook on your inventory file:

```
# ansible-playbook -i inventory_file path/custom-playbook.yml
```

Verification

- Verify that the managed node has the correct configuration by opening the SSH configuration file in a text editor, for example:

```
# vi ~root/.ssh/config
```

After application of the example playbook shown above, the configuration file should have the following content:

```
# Ansible managed
Compression yes
ControlMaster auto
ControlPath ~/.ssh/.cm%C
ForwardX11 no
GSSAPIAuthentication no
Host example
  Hostname example.com
  User user1
```

13.5. USING THE SSHD SYSTEM ROLE FOR NON-EXCLUSIVE

CONFIGURATION

Normally, applying the **sshd** System Role overwrites the entire configuration. This may be problematic if you have previously adjusted the configuration, for example with a different System Role or playbook. To apply the **sshd** System Role for only selected configuration options while keeping other options in place, you can use the non-exclusive configuration.

In RHEL 8 and earlier, you can apply the non-exclusive configuration with a configuration snippet.

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **sshd** System Role.
- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.
  On the control node:
  - The **ansible-core** package is installed.
  - An inventory file which lists the managed nodes.
  - A playbook for a different RHEL System Role.

Procedure

1. Add a configuration snippet with the **sshd_config_namespace** variable to the playbook:

```
---
- hosts: all
  tasks:
  - name: <Configure SSHD to accept some useful environment variables>
    include_role:
      name: rhel-system-roles.sshd
    vars:
      sshd_config_namespace: <my-application>
      sshd:
        # Environment variables to accept
        AcceptEnv:
          LANG
          LS_COLORS
          EDITOR
```

   When you apply the playbook to the inventory, the role adds the following snippet, if not already present, to the **/etc/ssh/sshd_config** file.

```
# BEGIN sshd system role managed block: namespace <my-application>
Match all
  AcceptEnv LANG LS_COLORS EDITOR
# END sshd system role managed block: namespace <my-application>
```

Verification

- Optional: Verify playbook syntax.

```
# ansible-playbook --syntax-check playbook.yml -i inventory_file
```

Additional resources

- **/usr/share/doc/rhel-system-roles/sshd/README.md** file.
- **ansible-playbook(1)** man page.

CHAPTER 14. CONFIGURING VPN CONNECTIONS WITH IPSEC BY USING THE VPN RHEL SYSTEM ROLE

With the **vpn** System Role, you can configure VPN connections on RHEL systems by using Red Hat Ansible Automation Platform. You can use it to set up host-to-host, network-to-network, VPN Remote Access Server, and mesh configurations.

For host-to-host connections, the role sets up a VPN tunnel between each pair of hosts in the list of **vpn_connections** using the default parameters, including generating keys as needed. Alternatively, you can configure it to create an opportunistic mesh configuration between all hosts listed. The role assumes that the names of the hosts under **hosts** are the same as the names of the hosts used in the Ansible inventory, and that you can use those names to configure the tunnels.

NOTE

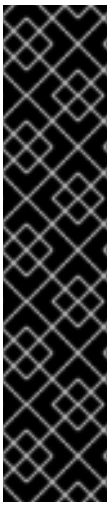
The **vpn** RHEL System Role currently supports only Libreswan, which is an IPsec implementation, as the VPN provider.

14.1. CREATING A HOST-TO-HOST VPN WITH IPSEC USING THE VPN SYSTEM ROLE

You can use the **vpn** System Role to configure host-to-host connections by running an Ansible playbook on the control node, which will configure all the managed nodes listed in an inventory file.

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **vpn** System Role.
- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.
  On the control node:
  - The **ansible-core** and **rhel-system-roles** packages are installed.



IMPORTANT

RHEL 8.0-8.5 provided access to a separate Ansible repository that contains Ansible Engine 2.9 for automation based on Ansible. Ansible Engine contains command-line utilities such as **ansible**, **ansible-playbook**, connectors such as **docker** and **podman**, and many plugins and modules. For information on how to obtain and install Ansible Engine, see the How to download and install Red Hat Ansible Engine Knowledgebase article.

RHEL 8.6 and 9.0 have introduced Ansible Core (provided as the **ansible-core** package), which contains the Ansible command-line utilities, commands, and a small set of built-in Ansible plugins. RHEL provides this package through the AppStream repository, and it has a limited scope of support. For more information, see the Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories Knowledgebase article.

- An inventory file which lists the managed nodes.

Procedure

1. Create a new ***playbook.yml*** file with the following content:

```
- name: Host to host VPN
  hosts: managed_node1, managed_node2
  roles:
    - rhel-system-roles.vpn
  vars:
    vpn_connections:
      - hosts:
          managed_node1:
          managed_node2:
```

This playbook configures the connection ***managed_node1*-to-*managed_node2*** using pre-shared key authentication with keys auto-generated by the system role.

2. Optional: Configure connections from managed hosts to external hosts that are not listed in the inventory file by adding the following section to the **vpn_connections** list of hosts:

```
    vpn_connections:
      - hosts:
          managed_node1:
          managed_node2:
          external_node:
            hostname: 192.0.2.2
```

This configures two additional connections: ***managed_node1*-to-*external_node*** and ***managed_node2*-to-*external_node***.

NOTE

The connections are configured only on the managed nodes and not on the external node.

1. Optional: You can specify multiple VPN connections for the managed nodes by using additional sections within **vpn_connections**, for example a control plane and a data plane:

```
- name: Multiple VPN
  hosts: managed_node1, managed_node2
  roles:
    - rhel-system-roles.vpn
  vars:
    vpn_connections:
      - name: control_plane_vpn
        hosts:
          managed_node1:
            hostname: 192.0.2.0 # IP for the control plane
          managed_node2:
            hostname: 192.0.2.1
      - name: data_plane_vpn
        hosts:
          managed_node1:
            hostname: 10.0.0.1 # IP for the data plane
          managed_node2:
            hostname: 10.0.0.2
```

2. Optional: You can modify the variables according to your preferences. For more details, see the **/usr/share/doc/rhel-system-roles/vpn/README.md** file.

3. Optional: Verify playbook syntax.

```
# ansible-playbook --syntax-check /path/to/file/playbook.yml -i /path/to/file/inventory_file
```

4. Run the playbook on your inventory file:

```
# ansible-playbook -i /path/to/file/inventory_file /path/to/file/playbook.yml
```

Verification

1. On the managed nodes, confirm that the connection is successfully loaded:

```
# ipsec status | grep connection.name
```

Replace *connection.name* with the name of the connection from this node, for example **managed_node1-to-managed_node2**.

NOTE

By default, the role generates a descriptive name for each connection it creates from the perspective of each system. For example, when creating a connection between **managed_node1** and **managed_node2**, the descriptive name of this connection on **managed_node1** is **managed_node1-to-managed_node2** but on **managed_node2** the connection is named **managed_node2-to-managed_node1**.

1. On the managed nodes, confirm that the connection is successfully started:

```
# ipsec trafficstatus | grep connection.name
```

2. Optional: If a connection did not successfully load, manually add the connection by entering the following command. This will provide more specific information indicating why the connection failed to establish:

```
# ipsec auto --add connection.name
```

NOTE

Any errors that may have occurred during the process of loading and starting the connection are reported in the logs, which can be found in **/var/log/pluto.log**. Because these logs are hard to parse, try to manually add the connection to obtain log messages from the standard output instead.

14.2. CREATING AN OPPORTUNISTIC MESH VPN CONNECTION WITH IPSEC BY USING THE VPN SYSTEM ROLE

You can use the **vpn** System Role to configure an opportunistic mesh VPN connection that uses certificates for authentication by running an Ansible playbook on the control node, which will configure all the managed nodes listed in an inventory file.

Authentication with certificates is configured by defining the **auth_method: cert** parameter in the playbook. The **vpn** System Role assumes that the IPsec Network Security Services (NSS) crypto library, which is defined in the **/etc/ipsec.d** directory, contains the necessary certificates. By default, the node name is used as the certificate nickname. In this example, this is **managed_node1**. You can define different certificate names by using the **cert_name** attribute in your inventory.

In the following example procedure, the control node, which is the system from which you will run the Ansible playbook, shares the same classless inter-domain routing (CIDR) number as both of the managed nodes (192.0.2.0/24) and has the IP address 192.0.2.7. Therefore, the control node falls under the private policy which is automatically created for CIDR 192.0.2.0/24.

To prevent SSH connection loss during the play, a clear policy for the control node is included in the list of policies. Note that there is also an item in the policies list where the CIDR is equal to default. This is because this playbook overrides the rule from the default policy to make it private instead of private-or-clear.

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **vpn** System Role.
    - On all the managed nodes, the NSS database in the **/etc/ipsec.d** directory contains all the certificates necessary for peer authentication. By default, the node name is used as the certificate nickname.
- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.
  On the control node:
    - The **ansible-core** and **rhel-system-roles** packages are installed.

IMPORTANT

RHEL 8.0-8.5 provided access to a separate Ansible repository that contains Ansible Engine 2.9 for automation based on Ansible. Ansible Engine contains command-line utilities such as **ansible**, **ansible-playbook**, connectors such as **docker** and **podman**, and many plugins and modules. For information on how to obtain and install Ansible Engine, see the How to download and install Red Hat Ansible Engine Knowledgebase article.

RHEL 8.6 and 9.0 have introduced Ansible Core (provided as the **ansible-core** package), which contains the Ansible command-line utilities, commands, and a small set of built-in Ansible plugins. RHEL provides this package through the AppStream repository, and it has a limited scope of support. For more information, see the Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories Knowledgebase article.

- An inventory file which lists the managed nodes.

Procedure

1. Create a new ***playbook.yml*** file with the following content:

```
- name: Mesh VPN
  hosts: managed_node1, managed_node2, managed_node3
  roles:
```

```
    - rhel-system-roles.vpn
  vars:
    vpn_connections:
      - opportunistic: true
        auth_method: cert
        policies:
          - policy: private
            cidr: default
          - policy: private-or-clear
            cidr: 198.51.100.0/24
          - policy: private
            cidr: 192.0.2.0/24
          - policy: clear
            cidr: 192.0.2.7/32
```

2. Optional: You can modify the variables according to your preferences. For more details, see the **/usr/share/doc/rhel-system-roles/vpn/README.md** file.

3. Optional: Verify playbook syntax.

```
# ansible-playbook --syntax-check playbook.yml
```

4. Run the playbook on your inventory file:

```
# ansible-playbook -i inventory_file /path/to/file/playbook.yml
```

14.3. ADDITIONAL RESOURCES

- For details about the parameters used in the **vpn** System Role and additional information about the role, see the **/usr/share/doc/rhel-system-roles/vpn/README.md** file.

- For details about the **ansible-playbook** command, see the **ansible-playbook(1)** man page.

CHAPTER 15. SETTING A CUSTOM CRYPTOGRAPHIC POLICY ACROSS SYSTEMS

As an administrator, you can use the System-wide **crypto_policies** RHEL System Role to quickly and consistently configure custom cryptographic policies across many different systems using Red Hat Ansible Automation Platform.

15.1. CRYPTO_POLICIES SYSTEM ROLE VARIABLES AND FACTS

In a **crypto_policies** System Role playbook, you can define the parameters for the **crypto_policies** configuration file according to your preferences and limitations.

If you do not configure any variables, the System Role does not configure the system and only reports the facts.

Selected variables for the crypto_policies System Role

crypto_policies_policy

Determines the cryptographic policy the System Role applies to the managed nodes. For details about the different crypto policies, see System-wide cryptographic policies .

crypto_policies_reload

If set to **yes**, the affected services, currently the **ipsec**, **bind**, and **sshd** services, reload after applying a crypto policy. Defaults to **yes**.

crypto_policies_reboot_ok

If set to **yes**, and a reboot is necessary after the System Role changes the crypto policy, it sets **crypto_policies_reboot_required** to **yes**. Defaults to **no**.

Facts set by the crypto_policies System Role

crypto_policies_active

Lists the currently selected policy.

crypto_policies_available_policies

Lists all available policies available on the system.

crypto_policies_available_subpolicies

Lists all available subpolicies available on the system.

Additional resources

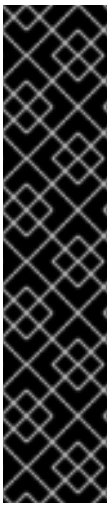
- Creating and setting a custom system-wide cryptographic policy .

15.2. SETTING A CUSTOM CRYPTOGRAPHIC POLICY USING THE CRYPTO_POLICIES SYSTEM ROLE

You can use the **crypto_policies** System Role to configure a large number of managed nodes consistently from a single control node.

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **crypto_policies** System Role.
- Access and permissions to a control node, which is a system from which Red Hat Ansible Core configures other systems.
  On the control node:
  - The **ansible-core** and **rhel-system-roles** packages are installed.



IMPORTANT

RHEL 8.0-8.5 provided access to a separate Ansible repository that contains Ansible Engine 2.9 for automation based on Ansible. Ansible Engine contains command-line utilities such as **ansible**, **ansible-playbook**, connectors such as **docker** and **podman**, and many plugins and modules. For information on how to obtain and install Ansible Engine, see the How to download and install Red Hat Ansible Engine Knowledgebase article.

RHEL 8.6 and 9.0 have introduced Ansible Core (provided as the **ansible-core** package), which contains the Ansible command-line utilities, commands, and a small set of built-in Ansible plugins. RHEL provides this package through the AppStream repository, and it has a limited scope of support. For more information, see the Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories Knowledgebase article.

- An inventory file which lists the managed nodes.

Procedure

1. Create a new ***playbook.yml*** file with the following content:

```
---
- hosts: all
  tasks:
  - name: Configure crypto policies
    include_role:
      name: rhel-system-roles.crypto_policies
    vars:
      - crypto_policies_policy: FUTURE
      - crypto_policies_reboot_ok: true
```

   You can replace the *FUTURE* value with your preferred crypto policy, for example: **DEFAULT**, **LEGACY**, and **FIPS:OSPP**.

   The **crypto_policies_reboot_ok: true** variable causes the system to reboot after the System Role changes the cryptographic policy.

   For more details, see crypto_policies System Role variables and facts .

2. Optional: Verify playbook syntax.

```
# ansible-playbook --syntax-check playbook.yml
```

3. Run the playbook on your inventory file:

```
# ansible-playbook -i inventory_file playbook.yml
```

-

Verification

1. On the control node, create another playbook named, for example, ***verify_playbook.yml***:

```
- hosts: all
  tasks:
 - name: Verify active crypto policy
   include_role:
     name: rhel-system-roles.crypto_policies

 - debug:
     var: crypto_policies_active
```

   This playbook does not change any configurations on the system, only reports the active policy on the managed nodes.

2. Run the playbook on the same inventory file:

```
# ansible-playbook -i inventory_file verify_playbook.yml

TASK [debug] ***************************
ok: [host] => {
    "crypto_policies_active": "FUTURE"
}
```

   The **"crypto_policies_active":** variable shows the policy active on the managed node.

15.3. ADDITIONAL RESOURCES

- **/usr/share/ansible/roles/rhel-system-roles.crypto_policies/README.md** file.
- **ansible-playbook(1)** man page.
- Installing RHEL System Roles .
- Applying a system role .

CHAPTER 16. USING THE NBDE_CLIENT AND NBDE_SERVER SYSTEM ROLES

16.1. INTRODUCTION TO THE NBDE_CLIENT AND NBDE_SERVER SYSTEM ROLES (CLEVIS AND TANG)

RHEL System Roles is a collection of Ansible roles and modules that provide a consistent configuration interface to remotely manage multiple RHEL systems.

RHEL 8.3 introduced Ansible roles for automated deployments of Policy-Based Decryption (PBD) solutions using Clevis and Tang. The **rhel-system-roles** package contains these system roles, related examples, and also the reference documentation.

The **nbde_client** System Role enables you to deploy multiple Clevis clients in an automated way. Note that the **nbde_client** role supports only Tang bindings, and you cannot use it for TPM2 bindings at the moment.

The **nbde_client** role requires volumes that are already encrypted using LUKS. This role supports to bind a LUKS-encrypted volume to one or more Network-Bound (NBDE) servers - Tang servers. You can either preserve the existing volume encryption with a passphrase or remove it. After removing the passphrase, you can unlock the volume only using NBDE. This is useful when a volume is initially encrypted using a temporary key or password that you should remove after you provision the system.

If you provide both a passphrase and a key file, the role uses what you have provided first. If it does not find any of these valid, it attempts to retrieve a passphrase from an existing binding.

PBD defines a binding as a mapping of a device to a slot. This means that you can have multiple bindings for the same device. The default slot is slot 1.

The **nbde_client** role provides also the **state** variable. Use the **present** value for either creating a new binding or updating an existing one. Contrary to a **clevis luks bind** command, you can use **state: present** also for overwriting an existing binding in its device slot. The **absent** value removes a specified binding.

Using the **nbde_client** System Role, you can deploy and manage a Tang server as part of an automated disk encryption solution. This role supports the following features:

- Rotating Tang keys
- Deploying and backing up Tang keys

Additional resources

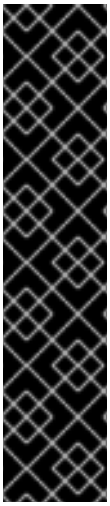
- For a detailed reference on Network-Bound Disk Encryption (NBDE) role variables, install the **rhel-system-roles** package, and see the **README.md** and **README.html** files in the **/usr/share/doc/rhel-system-roles/nbde_client/** and **/usr/share/doc/rhel-system-roles/nbde_server/** directories.
- For example system-roles playbooks, install the **rhel-system-roles** package, and see the **/usr/share/ansible/roles/rhel-system-roles.nbde_server/examples/** directories.
- For more information on RHEL System Roles, see Introduction to RHEL System Roles

16.2. USING THE NBDE_SERVER SYSTEM ROLE FOR SETTING UP MULTIPLE TANG SERVERS

Follow the steps to prepare and apply an Ansible playbook containing your Tang server settings.

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **nbde_server** System Role.
- Access and permissions to a control node, which is a system from which Red Hat Ansible Core configures other systems.
  On the control node:
  - The **ansible-core** and **rhel-system-roles** packages are installed.



IMPORTANT

RHEL 8.0-8.5 provided access to a separate Ansible repository that contains Ansible Engine 2.9 for automation based on Ansible. Ansible Engine contains command-line utilities such as **ansible**, **ansible-playbook**, connectors such as **docker** and **podman**, and many plugins and modules. For information on how to obtain and install Ansible Engine, see the How to download and install Red Hat Ansible Engine Knowledgebase article.

RHEL 8.6 and 9.0 have introduced Ansible Core (provided as the **ansible-core** package), which contains the Ansible command-line utilities, commands, and a small set of built-in Ansible plugins. RHEL provides this package through the AppStream repository, and it has a limited scope of support. For more information, see the Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories Knowledgebase article.

- An inventory file which lists the managed nodes.

Procedure

1. Prepare your playbook containing settings for Tang servers. You can either start from the scratch, or use one of the example playbooks from the **/usr/share/ansible/roles/rhel-system-roles.nbde_server/examples/** directory.

```
# cp /usr/share/ansible/roles/rhel-system-roles.nbde_server/examples/simple_deploy.yml
./my-tang-playbook.yml
```

2. Edit the playbook in a text editor of your choice, for example:

```
# vi my-tang-playbook.yml
```

3. Add the required parameters. The following example playbook ensures deploying of your Tang server and a key rotation:

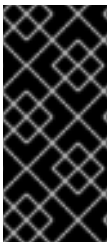
```
---
- hosts: all

  vars:
```

```
    nbde_server_rotate_keys: yes

  roles:
    - rhel-system-roles.nbde_server
```

4. Apply the finished playbook:

```
# ansible-playbook -i inventory-file my-tang-playbook.yml
```

Where: * ***inventory-file*** is the inventory file. * ***logging-playbook.yml*** is the playbook you use.



IMPORTANT

To ensure that networking for a Tang pin is available during early boot by using the **grubby** tool on the systems where Clevis is installed:

```
# grubby --update-kernel=ALL --args="rd.neednet=1"
```

Additional resources

- For more information, install the **rhel-system-roles** package, and see the **/usr/share/doc/rhel-system-roles/nbde_server/** and **usr/share/ansible/roles/rhel-system-roles.nbde_server/** directories.

16.3. USING THE NBDE_CLIENT SYSTEM ROLE FOR SETTING UP MULTIPLE CLEVIS CLIENTS

Follow the steps to prepare and apply an Ansible playbook containing your Clevis client settings.

NOTE

The **nbde_client** System Role supports only Tang bindings. This means that you cannot use it for TPM2 bindings at the moment.

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **nbde_client** System Role.
- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.
- The Ansible Core package is installed on the control machine.
- The **rhel-system-roles** package is installed on the system from which you want to run the playbook.

Procedure

1. Prepare your playbook containing settings for Clevis clients. You can either start from the scratch, or use one of the example playbooks from the **/usr/share/ansible/roles/rhel-system-roles.nbde_client/examples/** directory.

```
# cp /usr/share/ansible/roles/rhel-system-roles.nbde_client/examples/high_availability.yml
./my-clevis-playbook.yml
```

2. Edit the playbook in a text editor of your choice, for example:

```
# vi my-clevis-playbook.yml
```

3. Add the required parameters. The following example playbook configures Clevis clients for automated unlocking of two LUKS-encrypted volumes by when at least one of two Tang servers is available:

```
---
- hosts: all

  vars:
    nbde_client_bindings:
      - device: /dev/rhel/root
        encryption_key_src: /etc/luks/keyfile
        servers:
          - http://server1.example.com
          - http://server2.example.com
      - device: /dev/rhel/swap
        encryption_key_src: /etc/luks/keyfile
        servers:
          - http://server1.example.com
          - http://server2.example.com

  roles:
    - rhel-system-roles.nbde_client
```

4. Apply the finished playbook:

```
# ansible-playbook -i host1,host2,host3 my-clevis-playbook.yml
```

IMPORTANT

To ensure that networking for a Tang pin is available during early boot by using the **grubby** tool on the system where Clevis is installed:

```
# grubby --update-kernel=ALL --args="rd.neednet=1"
```

Additional resources

- For details about the parameters and additional information about the NBDE Client System Role, install the **rhel-system-roles** package, and see the **/usr/share/doc/rhel-system-roles/nbde_client/** and **/usr/share/ansible/roles/rhel-system-roles.nbde_client/** directories.

CHAPTER 17. REQUESTING CERTIFICATES USING RHEL SYSTEM ROLES

With the **certificate** System Role, you can use Red Hat Ansible Core to issue and manage certificates.

This chapter covers the following topics:

- The **certificate** System Role
- Requesting a new self-signed certificate using the **certificate** System Role
- Requesting a new certificate from IdM CA using the **certificate** System Role

17.1. THE CERTIFICATE SYSTEM ROLE

Using the **certificate** System Role, you can manage issuing and renewing TLS and SSL certificates using Ansible Core.

The role uses **certmonger** as the certificate provider, and currently supports issuing and renewing self-signed certificates and using the IdM integrated certificate authority (CA).

You can use the following variables in your Ansible playbook with the **certificate** System Role:

certificate_wait

to specify if the task should wait for the certificate to be issued.

certificate_requests

to represent each certificate to be issued and its parameters.

Additional resources

- See the **/usr/share/ansible/roles/rhel-system-roles.certificate/README.md** file.
- Preparing a control node and managed nodes to use RHEL System Roles

17.2. REQUESTING A NEW SELF-SIGNED CERTIFICATE USING THE CERTIFICATE SYSTEM ROLE

With the **certificate** System Role, you can use Ansible Core to issue self-signed certificates.

This process uses the **certmonger** provider and requests the certificate through the **getcert** command.

NOTE

By default, **certmonger** automatically tries to renew the certificate before it expires. You can disable this by setting the **auto_renew** parameter in the Ansible playbook to **no**.

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.

Procedure

1. *Optional:* Create an inventory file, for example **inventory.file**:

```
$ *touch inventory.file*
```

2. Open your inventory file and define the hosts on which you want to request the certificate, for example:

```
[webserver]
server.idm.example.com
```

3. Create a playbook file, for example **request-certificate.yml**:

   - Set **hosts** to include the hosts on which you want to request the certificate, such as **webserver**.
   - Set the **certificate_requests** variable to include the following:
     - Set the **name** parameter to the desired name of the certificate, such as **mycert**.
     - Set the **dns** parameter to the domain to be included in the certificate, such as ***.example.com**.
     - Set the **ca** parameter to **self-sign**.
   - Set the **rhel-system-roles.certificate** role under **roles**.
     This is the playbook file for this example:

```
---
- hosts: webserver

  vars:
    certificate_requests:
      - name: mycert
        dns: "*.example.com"
        ca: self-sign

  roles:
    - rhel-system-roles.certificate
```

4. Save the file.

5. Run the playbook:

```
$ *ansible-playbook -i inventory.file request-certificate.yml*
```

Additional resources

- See the **/usr/share/ansible/roles/rhel-system-roles.certificate/README.md** file.
- See the **ansible-playbook(1)** man page.

17.3. REQUESTING A NEW CERTIFICATE FROM IDM CA USING THE CERTIFICATE SYSTEM ROLE

With the **certificate** System Role, you can use **anible-core** to issue certificates while using an IdM server with an integrated certificate authority (CA). Therefore, you can efficiently and consistently manage the certificate trust chain for multiple systems when using IdM as the CA.

This process uses the **certmonger** provider and requests the certificate through the **getcert** command.

NOTE

By default, **certmonger** automatically tries to renew the certificate before it expires. You can disable this by setting the **auto_renew** parameter in the Ansible playbook to **no**.

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.

Procedure

1. *Optional:* Create an inventory file, for example **inventory.file**:

```
$ *touch inventory.file*
```

2. Open your inventory file and define the hosts on which you want to request the certificate, for example:

```
[webserver]
server.idm.example.com
```

3. Create a playbook file, for example **request-certificate.yml**:
   - Set **hosts** to include the hosts on which you want to request the certificate, such as **webserver**.
   - Set the **certificate_requests** variable to include the following:
     - Set the **name** parameter to the desired name of the certificate, such as **mycert**.
     - Set the **dns** parameter to the domain to be included in the certificate, such as **www.example.com**.
     - Set the **principal** parameter to specify the Kerberos principal, such as **HTTP/www.example.com@EXAMPLE.COM**.
     - Set the **ca** parameter to **ipa**.
   - Set the **rhel-system-roles.certificate** role under **roles**.
     This is the playbook file for this example:

```
---
- hosts: webserver
```

```
vars:
  certificate_requests:
    - name: mycert
      dns: www.example.com
      principal: HTTP/www.example.com@EXAMPLE.COM
      ca: ipa

roles:
  - rhel-system-roles.certificate
```

4. Save the file.
5. Run the playbook:

```
$ *ansible-playbook -i inventory.file request-certificate.yml*
```

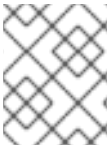
Additional resources

- See the **/usr/share/ansible/roles/rhel-system-roles.certificate/README.md** file.
- See the **ansible-playbook(1)** man page.

17.4. SPECIFYING COMMANDS TO RUN BEFORE OR AFTER CERTIFICATE ISSUANCE USING THE CERTIFICATE SYSTEM ROLE

With the **certificate** Role, you can use Ansible Core to execute a command before and after a certificate is issued or renewed.

In the following example, the administrator ensures stopping the **httpd** service before a self-signed certificate for **www.example.com** is issued or renewed, and restarting it afterwards.



NOTE

By default, **certmonger** automatically tries to renew the certificate before it expires. You can disable this by setting the **auto_renew** parameter in the Ansible playbook to **no**.

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.

Procedure

1. *Optional:* Create an inventory file, for example **inventory.file**:

```
$ *touch inventory.file*
```

2. Open your inventory file and define the hosts on which you want to request the certificate, for example:

```
[webserver]
server.idm.example.com
```

3. Create a playbook file, for example **request-certificate.yml**:
   - Set **hosts** to include the hosts on which you want to request the certificate, such as **webserver**.
   - Set the **certificate_requests** variable to include the following:
     - Set the **name** parameter to the desired name of the certificate, such as **mycert**.
     - Set the **dns** parameter to the domain to be included in the certificate, such as **www.example.com**.
     - Set the **ca** parameter to the CA you want to use to issue the certificate, such as **self-sign**.
     - Set the **run_before** parameter to the command you want to execute before this certificate is issued or renewed, such as **systemctl stop httpd.service**.
     - Set the **run_after** parameter to the command you want to execute after this certificate is issued or renewed, such as **systemctl start httpd.service**.
   - Set the **rhel-system-roles.certificate** role under **roles**.
     This is the playbook file for this example:

```
---
- hosts: webserver
  vars:
    certificate_requests:
      - name: mycert
        dns: www.example.com
        ca: self-sign
        run_before: systemctl stop httpd.service
        run_after: systemctl start httpd.service

  roles:
    - rhel-system-roles.certificate
```

4. Save the file.
5. Run the playbook:

```
$ *ansible-playbook -i inventory.file request-certificate.yml*
```

Additional resources

- See the **/usr/share/ansible/roles/rhel-system-roles.certificate/README.md** file.
- See the **ansible-playbook(1)** man page.

CHAPTER 18. CONFIGURING KDUMP USING RHEL SYSTEM ROLES

To manage kdump using Ansible, you can use the **kdump** role, which is one of the RHEL System Roles available in RHEL 8.

Using the **kdump** role enables you to specify where to save the contents of the system's memory for later analysis.

For more information about RHEL System Roles and how to apply them, see Introduction to RHEL System Roles.

18.1. THE KDUMP RHEL SYSTEM ROLE

The **kdump** System Role enables you to set basic kernel dump parameters on multiple systems.

18.2. KDUMP ROLE PARAMETERS

The parameters used for the **kdump** RHEL System Roles are:

Role Variable	Description
kdump_path	The path to which **vmcore** is written. If **kdump_target** is not null, path is relative to that dump target. Otherwise, it must be an absolute path in the root file system.

Additional resources

- The makedumpfile(8) man page.
- For details about the parameters used in **kdump** and additional information about the **kdump** System Role, see the **/usr/share/ansible/roles/rhel-system-roles.tlog/README.md** file.

18.3. CONFIGURING KDUMP USING RHEL SYSTEM ROLES

You can set basic kernel dump parameters on multiple systems using the **kdump** System Role by running an Ansible playbook.

WARNING

The **kdump** role replaces the kdump configuration of the managed hosts entirely by replacing the **/etc/kdump.conf** file. Additionally, if the **kdump** role is applied, all previous **kdump** settings are also replaced, even if they are not specified by the role variables, by replacing the **/etc/sysconfig/kdump** file.

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.
- You have an inventory file which lists the systems on which you want to deploy **kdump**.

Procedure

1. Create a new ***playbook.yml*** file with the following content:

```
---
- hosts: kdump-test
  vars:
    kdump_path: /var/crash
  roles:
    - rhel-system-roles.kdump
```

2. Optional: Verify playbook syntax.

```
# ansible-playbook --syntax-check playbook.yml
```

3. Run the playbook on your inventory file:

```
# ansible-playbook -i inventory_file /path/to/file/playbook.yml
```

Additional resources

- For a detailed reference on **kdump** role variables, see the README.md or README.html files in the /usr/share/doc/rhel-system-roles/kdump directory.
- See Preparing the control node and managed nodes to use RHEL System Roles
- Documentation installed with the **rhel-system-roles** package **/usr/share/ansible/roles/rhel-system-roles.kdump/README.html**

CHAPTER 19. MANAGING LOCAL STORAGE USING RHEL SYSTEM ROLES

To manage LVM and local file systems (FS) using Ansible, you can use the **storage** role, which is one of the RHEL System Roles available in RHEL 8.

Using the **storage** role enables you to automate administration of file systems on disks and logical volumes on multiple machines and across all versions of RHEL starting with RHEL 7.7.

For more information about RHEL System Roles and how to apply them, see Introduction to RHEL System Roles.

19.1. INTRODUCTION TO THE STORAGE RHEL SYSTEM ROLE

The **storage** role can manage:

- File systems on disks which have not been partitioned
- Complete LVM volume groups including their logical volumes and file systems
- MD RAID volumes and their file systems

With the **storage** role, you can perform the following tasks:

- Create a file system
- Remove a file system
- Mount a file system
- Unmount a file system
- Create LVM volume groups
- Remove LVM volume groups
- Create logical volumes
- Remove logical volumes
- Create RAID volumes
- Remove RAID volumes
- Create LVM volume groups with RAID
- Remove LVM volume groups with RAID
- Create encrypted LVM volume groups
- Create LVM logical volumes with RAID

19.2. PARAMETERS THAT IDENTIFY A STORAGE DEVICE IN THE STORAGE RHEL SYSTEM ROLE

Your **storage** role configuration affects only the file systems, volumes, and pools that you list in the following variables.

storage_volumes

List of file systems on all unpartitioned disks to be managed.
storage_volumes can also include **raid** volumes.

Partitions are currently unsupported.

storage_pools

List of pools to be managed.
Currently the only supported pool type is LVM. With LVM, pools represent volume groups (VGs). Under each pool there is a list of volumes to be managed by the role. With LVM, each volume corresponds to a logical volume (LV) with a file system.

19.3. EXAMPLE ANSIBLE PLAYBOOK TO CREATE AN XFS FILE SYSTEM ON A BLOCK DEVICE

This section provides an example Ansible playbook. This playbook applies the **storage** role to create an XFS file system on a block device using the default parameters.

WARNING

The **storage** role can create a file system only on an unpartitioned, whole disk or a logical volume (LV). It cannot create the file system on a partition.

Example 19.1. A playbook that creates XFS on /dev/sdb

```
---
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: xfs
  roles:
    - rhel-system-roles.storage
```

- The volume name (***barefs*** in the example) is currently arbitrary. The **storage** role identifies the volume by the disk device listed under the **disks:** attribute.
- You can omit the **fs_type: xfs** line because XFS is the default file system in RHEL 8.

- To create the file system on an LV, provide the LVM setup under the **disks:** attribute, including the enclosing volume group. For details, see Example Ansible playbook to manage logical volumes.
  Do not provide the path to the LV device.

Additional resources

- The **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file.

19.4. EXAMPLE ANSIBLE PLAYBOOK TO PERSISTENTLY MOUNT A FILE SYSTEM

This section provides an example Ansible playbook. This playbook applies the **storage** role to immediately and persistently mount an XFS file system.

Example 19.2. A playbook that mounts a file system on /dev/sdb to /mnt/data

```
---
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: xfs
        mount_point: /mnt/data
  roles:
    - rhel-system-roles.storage
```

- This playbook adds the file system to the **/etc/fstab** file, and mounts the file system immediately.
- If the file system on the **/dev/sdb** device or the mount point directory do not exist, the playbook creates them.

Additional resources

- The **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file.

19.5. EXAMPLE ANSIBLE PLAYBOOK TO MANAGE LOGICAL VOLUMES

This section provides an example Ansible playbook. This playbook applies the **storage** role to create an LVM logical volume in a volume group.

Example 19.3. A playbook that creates a mylv logical volume in the myvg volume group

```
- hosts: all
  vars:
```

```
  storage_pools:
    - name: myvg
      disks:
        - sda
        - sdb
        - sdc
      volumes:
        - name: mylv
          size: 2G
          fs_type: ext4
          mount_point: /mnt/data
roles:
  - rhel-system-roles.storage
```

- The **myvg** volume group consists of the following disks:
  - /**dev**/**sda**
  - /**dev**/**sdb**
  - /**dev**/**sdc**
- If the **myvg** volume group already exists, the playbook adds the logical volume to the volume group.
- If the **myvg** volume group does not exist, the playbook creates it.
- The playbook creates an Ext4 file system on the **mylv** logical volume, and persistently mounts the file system at /**mnt**.

Additional resources

- The /**usr**/**share**/**ansible**/**roles**/**rhel-system-roles.storage**/**README.md** file.

19.6. EXAMPLE ANSIBLE PLAYBOOK TO ENABLE ONLINE BLOCK DISCARD

This section provides an example Ansible playbook. This playbook applies the **storage** role to mount an XFS file system with online block discard enabled.

Example 19.4. A playbook that enables online block discard on /mnt/data/

```
---
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: xfs
        mount_point: /mnt/data
```

```
      mount_options: discard
roles:
  - rhel-system-roles.storage
```

Additional resources

- Example Ansible playbook to persistently mount a file system
- The **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file.

19.7. EXAMPLE ANSIBLE PLAYBOOK TO CREATE AND MOUNT AN EXT4 FILE SYSTEM

This section provides an example Ansible playbook. This playbook applies the **storage** role to create and mount an Ext4 file system.

Example 19.5. A playbook that creates Ext4 on /dev/sdb and mounts it at /mnt/data

```
---
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: ext4
        fs_label: label-name
        mount_point: /mnt/data
  roles:
    - rhel-system-roles.storage
```

- The playbook creates the file system on the **/dev/sdb** disk.
- The playbook persistently mounts the file system at the ***/mnt/data*** directory.
- The label of the file system is ***label-name***.

Additional resources

- The **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file.

19.8. EXAMPLE ANSIBLE PLAYBOOK TO CREATE AND MOUNT AN EXT3 FILE SYSTEM

This section provides an example Ansible playbook. This playbook applies the **storage** role to create and mount an Ext3 file system.

Example 19.6. A playbook that creates Ext3 on/dev/sdb and mounts it at/mnt/data

```
---
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: ext3
        fs_label: label-name
        mount_point: /mnt/data
  roles:
    - rhel-system-roles.storage
```

- The playbook creates the file system on the **/dev/sdb** disk.
- The playbook persistently mounts the file system at the ***/mnt/data*** directory.
- The label of the file system is ***label-name***.

Additional resources

- The **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file.

19.9. EXAMPLE ANSIBLE PLAYBOOK TO RESIZE AN EXISTING EXT4 OR EXT3 FILE SYSTEM USING THE STORAGE RHEL SYSTEM ROLE

This section provides an example Ansible playbook. This playbook applies the **storage** role to resize an existing Ext4 or Ext3 file system on a block device.

Example 19.7. A playbook that set up a single volume on a disk

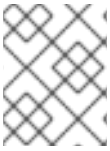
```
---
- name: Create a disk device mounted on /opt/barefs
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - /dev/sdb
        size: 12 GiB
        fs_type: ext4
        mount_point: /opt/barefs
  roles:
    - rhel-system-roles.storage
```

- If the volume in the previous example already exists, to resize the volume, you need to run the same playbook, just with a different value for the parameter **size**. For example:

Example 19.8. A playbook that resizes ext4 on /dev/sdb

```
---
- name: Create a disk device mounted on /opt/barefs
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - /dev/sdb
        size: 10 GiB
        fs_type: ext4
        mount_point: /opt/barefs
  roles:
    - rhel-system-roles.storage
```

- The volume name (barefs in the example) is currently arbitrary. The Storage role identifies the volume by the disk device listed under the disks: attribute.



NOTE

Using the **Resizing** action in other file systems can destroy the data on the device you are working on.

Additional resources

- The **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file.

19.10. EXAMPLE ANSIBLE PLAYBOOK TO RESIZE AN EXISTING FILE SYSTEM ON LVM USING THE STORAGE RHEL SYSTEM ROLE

This section provides an example Ansible playbook. This playbook applies the **storage** RHEL System Role to resize an LVM logical volume with a file system.

WARNING

Using the **Resizing** action in other file systems can destroy the data on the device you are working on.

Example 19.9. A playbook that resizes existing mylv1 and myvl2 logical volumes in the myvg volume group

```
---

- hosts: all
  vars:
    storage_pools:
      - name: myvg
```

```
      disks:
        - /dev/sda
        - /dev/sdb
        - /dev/sdc
      volumes:
        - name: mylv1
          size: 10 GiB
          fs_type: ext4
          mount_point: /opt/mount1
        - name: mylv2
          size: 50 GiB
          fs_type: ext4
          mount_point: /opt/mount2

- name: Create LVM pool over three disks
  incude_role:
    name: rhel-system-roles.storage
```

- This playbook resizes the following existing file systems:
  - The Ext4 file system on the **mylv1** volume, which is mounted at **/opt/mount1**, resizes to 10 GiB.
  - The Ext4 file system on the **mylv2** volume, which is mounted at **/opt/mount2**, resizes to 50 GiB.

Additional resources

- The **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file.

19.11. EXAMPLE ANSIBLE PLAYBOOK TO CREATE A SWAP VOLUME USING THE STORAGE RHEL SYSTEM ROLE

This section provides an example Ansible playbook. This playbook applies the **storage** role to create a swap volume, if it does not exist, or to modify the swap volume, if it already exist, on a block device using the default parameters.

Example 19.10. A playbook that creates or modify an existing XFS on /dev/sdb

```
---
- name: Create a disk device with swap
- hosts: all
  vars:
    storage_volumes:
      - name: swap_fs
        type: disk
        disks:
          - /dev/sdb
 size: 15 GiB
        fs_type: swap
  roles:
    - rhel-system-roles.storage
```

- The volume name (***swap_fs*** in the example) is currently arbitrary. The **storage** role identifies the volume by the disk device listed under the **disks:** attribute.

Additional resources

- The **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file.

19.12. CONFIGURING A RAID VOLUME USING THE STORAGE RHEL SYSTEM ROLE

With the **storage** System Role, you can configure a RAID volume on RHEL using Red Hat Ansible Automation Platform. In this section you will learn how to set up an Ansible playbook with the available parameters to configure a RAID volume to suit your requirements.

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.
- You have an inventory file detailing the systems on which you want to deploy a RAID volume using the **storage** System Role.

Procedure

1. Create a new ***playbook.yml*** file with the following content:

```
- hosts: all
  vars:
    storage_safe_mode: false
    storage_volumes:
      - name: data
        type: raid
        disks: [sdd, sde, sdf, sdg]
        raid_level: raid0
        raid_chunk_size: 32 KiB
        mount_point: /mnt/data
        state: present
  roles:
    - name: rhel-system-roles.storage
```

WARNING

Device names can change in certain circumstances; for example, when you add a new disk to a system. Therefore, to prevent data loss, we do not recommend using specific disk names in the playbook.

2. Optional. Verify playbook syntax.

```
# ansible-playbook --syntax-check playbook.yml
```

3. Run the playbook on your inventory file:

```
# ansible-playbook -i inventory.file /path/to/file/playbook.yml
```

Additional resources

- Managing RAID.
- The **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file.

19.13. CONFIGURING AN LVM POOL WITH RAID USING THE STORAGE RHEL SYSTEM ROLE

With the **storage** System Role, you can configure an LVM pool with RAID on RHEL using Red Hat Ansible Automation Platform. In this section you will learn how to set up an Ansible playbook with the available parameters to configure an LVM pool with RAID.

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.
- You have an inventory file detailing the systems on which you want to configure an LVM pool with RAID using the **storage** System Role.

Procedure

1. Create a new ***playbook.yml*** file with the following content:

```
- hosts: all
  vars:
    storage_safe_mode: false
    storage_pools:
      - name: my_pool
        type: lvm
        disks: [sdh, sdi]
        raid_level: raid1
        volumes:
          - name: my_pool
            size: "1 GiB"
            mount_point: "/mnt/app/shared"
            fs_type: xfs
            state: present
  roles:
    - name: rhel-system-roles.storage
```

NOTE

To create an LVM pool with RAID, you must specify the RAID type using the **raid_level** parameter.

2. Optional. Verify playbook syntax.

```
# ansible-playbook --syntax-check playbook.yml
```

3. Run the playbook on your inventory file:

```
# ansible-playbook -i inventory.file /path/to/file/playbook.yml
```

Additional resources

- Managing RAID.
- The **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file.

19.14. EXAMPLE ANSIBLE PLAYBOOK TO COMPRESS AND DEDUPLICATE A VDO VOLUME ON LVM USING THE STORAGE RHEL SYSTEM ROLE

This section provides an example Ansible playbook. This playbook applies the **storage** RHEL System Role to enable compression and deduplication of Logical Volumes (LVM) using Virtual Data Optimizer (VDO).

Example 19.11. A playbook that creates a mylv1 LVM VDO volume in the myvg volume group

```
---
- name: Create LVM VDO volume under volume group 'myvg'
  hosts: all
  roles:
    -rhel-system-roles.storage
  vars:
    storage_pools:
     - name: myvg
       disks:
         - /dev/sdb
       volumes:
         - name: mylv1
           compression: true
           deduplication: true
           vdo_pool_size: 10 GiB
           size: 30 GiB
           mount_point: /mnt/app/shared
```

In this example, the **compression** and **deduplication** pools are set to true, which specifies that the VDO is used. The following describes the usage of these parameters:

- The **deduplication** is used to deduplicate the duplicated data stored on the storage volume.

- The compression is used to compress the data stored on the storage volume, which results in more storage capacity.
- The vdo_pool_size specifies the actual size the volume takes on the device. The virtual size of VDO volume is set by the **size** parameter. NOTE: Because of the Storage role use of LVM VDO, only one volume per pool can use the compression and deduplication.

19.15. CREATING A LUKS ENCRYPTED VOLUME USING THE STORAGE RHEL SYSTEM ROLE

You can use the **storage** role to create and configure a volume encrypted with LUKS by running an Ansible playbook.

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **crypto_policies** System Role.
- Access and permissions to a control node, which is a system from which Red Hat Ansible Core configures other systems.
  On the control node:
  - The **ansible-core** and **rhel-system-roles** packages are installed.

IMPORTANT

RHEL 8.0-8.5 provided access to a separate Ansible repository that contains Ansible Engine 2.9 for automation based on Ansible. Ansible Engine contains command-line utilities such as **ansible**, **ansible-playbook**, connectors such as **docker** and **podman**, and many plugins and modules. For information on how to obtain and install Ansible Engine, see the How to download and install Red Hat Ansible Engine Knowledgebase article.

RHEL 8.6 and 9.0 have introduced Ansible Core (provided as the **ansible-core** package), which contains the Ansible command-line utilities, commands, and a small set of built-in Ansible plugins. RHEL provides this package through the AppStream repository, and it has a limited scope of support. For more information, see the Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories Knowledgebase article.

- An inventory file which lists the managed nodes.

Procedure

1. Create a new ***playbook.yml*** file with the following content:

```
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: xfs
        fs_label: label-name
```

```
      mount_point: /mnt/data
      encryption: true
      encryption_password: your-password
  roles:
   - rhel-system-roles.storage
```

2. Optional: Verify playbook syntax:

```
# ansible-playbook --syntax-check playbook.yml
```

3. Run the playbook on your inventory file:

```
# ansible-playbook -i inventory.file /path/to/file/playbook.yml
```

Additional resources

- Encrypting block devices using LUKS
- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file

19.16. EXAMPLE ANSIBLE PLAYBOOK TO EXPRESS POOL VOLUME SIZES AS PERCENTAGE USING THE STORAGE RHEL SYSTEM ROLE

This section provides an example Ansible playbook. This playbook applies the **storage** System Role to enable you to express Logical Manager Volumes (LVM) volume sizes as a percentage of the pool's total size.

Example 19.12. A playbook that express volume sizes as a percentage of the pool's total size

```
---
- name: Express volume sizes as a percentage of the pool's total size
  hosts: all
  roles
    - rhel-system-roles.storage
  vars:
    storage_pools:
    - name: myvg
      disks:
        - /dev/sdb
      volumes:
        - name: data
          size: 60%
          mount_point: /opt/mount/data
        - name: web
          size: 30%
          mount_point: /opt/mount/web
        - name: cache
          size: 10%
          mount_point: /opt/cache/mount
```

This example specifies the size of LVM volumes as a percentage of the pool size, for example: "60%". Additionally, you can also specify the size of LVM volumes as a percentage of the pool size in a human-readable size of the file system, for example, "10g" or "50 GiB".

19.17. ADDITIONAL RESOURCES

- **/usr/share/doc/rhel-system-roles/storage/**
- **/usr/share/ansible/roles/rhel-system-roles.storage/**

CHAPTER 20. CONFIGURING TIME SYNCHRONIZATION USING RHEL SYSTEM ROLES

With the **timesync** RHEL System Role, you can manage time synchronization on multiple target machines on RHEL using Red Hat Ansible Automation Platform.

20.1. THE TIMESYNC RHEL SYSTEM ROLE

You can manage time synchronization on multiple target machines using the **timesync** RHEL System Role.

The **timesync** role installs and configures an NTP or PTP implementation to operate as an NTP client or PTP replica in order to synchronize the system clock with NTP servers or grandmasters in PTP domains.

Note that using the **timesync** role also facilitates the migration to chrony, because you can use the same playbook on all versions of Red Hat Enterprise Linux starting with RHEL 6 regardless of whether the system uses **ntp** or **chrony** to implement the NTP protocol.

20.2. APPLYING THE TIMESYNC SYSTEM ROLE FOR A SINGLE POOL OF SERVERS

The following example shows how to apply the **timesync** role in a situation with just one pool of servers.

WARNING

The **timesync** role replaces the configuration of the given or detected provider service on the managed host. Previous settings are lost, even if they are not specified in the role variables. The only preserved setting is the choice of provider if the **timesync_ntp_provider** variable is not defined.

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.
- You have an inventory file which lists the systems on which you want to deploy **timesync** System Role.

Procedure

1. Create a new ***playbook.yml*** file with the following content:

```
---
- hosts: timesync-test
  vars:
    timesync_ntp_servers:
```

```
    - hostname: 2.rhel.pool.ntp.org
      pool: yes
      iburst: yes
  roles:
    - rhel-system-roles.timesync
```

2. Optional: Verify playbook syntax.

```
# ansible-playbook --syntax-check playbook.yml
```

3. Run the playbook on your inventory file:

```
# ansible-playbook -i inventory_file /path/to/file/playbook.yml
```

20.3. APPLYING THE TIMESYNC SYSTEM ROLE ON CLIENT SERVERS

You can use the **timesync** role to enable Network Time Security (NTS) on NTP clients. Network Time Security (NTS) is an authentication mechanism specified for Network Time Protocol (NTP). It verifies that NTP packets exchanged between the server and client are not altered.

WARNING

The **timesync** role replaces the configuration of the given or detected provider service on the managed host. Previous settings are lost even if they are not specified in the role variables. The only preserved setting is the choice of provider if the **timesync_ntp_provider** variable is not defined.

Prerequisites

- You do not have to have Red Hat Ansible Automation Platform installed on the systems on which you want to deploy the **timesync** solution.
- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.
- You have an inventory file which lists the systems on which you want to deploy the **timesync** System Role.
- The **chrony** NTP provider version is 4.0 or later.

Procedure

1. Create a ***playbook.yml*** file with the following content:

```
---
- hosts: timesync-test
  vars:
    timesync_ntp_servers:
      - hostname: ptbtime1.ptb.de
```

```
      iburst: yes
      nts: yes
  roles:
    - rhel-system-roles.timesync
```

ptbtime1.ptb.de is an example of public server. You may want to use a different public server or your own server.

2. Optional: Verify playbook syntax.

```
# ansible-playbook --syntax-check playbook.yml
```

3. Run the playbook on your inventory file:

```
# ansible-playbook -i inventory_file /path/to/file/playbook.yml
```

Verification

1. Perform a test on the client machine:

```
# chronyc -N authdata

Name/IP address         Mode KeyID Type KLen Last Atmp  NAK Cook CLen
==============================================================================
ptbtime1.ptb.de         NTS     1   15  256  157    0    0    8  100
```

2. Check that the number of reported cookies is larger than zero.

Additional resources

- **chrony.conf(5)** man page

20.4. TIMESYNC SYSTEM ROLES VARIABLES

You can pass the following variable to the **timesync** role:

- **timesync_ntp_servers**:

Role variable settings	Description
hostname: host.example.com	Hostname or address of the server
minpoll: *number*	Minimum polling interval. Default: 6
maxpoll: *number*	Maximum polling interval. Default: 10
iburst: yes	Flag enabling fast initial synchronization. Default: no
pool: yes	Flag indicating that each resolved address of the hostname is a separate NTP server. Default: no

Role variable settings	Description
nts: yes	Flag to enable Network Time Security (NTS). Default: no. Supported only with chrony >= 4.0.

Additional resources

- For a detailed reference on **timesync** role variables, install the rhel-system-roles package, and see the README.md or README.html files in the **/usr/share/doc/rhel-system-roles/timesync** directory.

CHAPTER 21. MONITORING PERFORMANCE USING RHEL SYSTEM ROLES

As a system administrator, you can use the **metrics** RHEL System Role with any Ansible Automation Platform control node to monitor the performance of a system.

21.1. INTRODUCTION TO THE METRICS SYSTEM ROLE

RHEL System Roles is a collection of Ansible roles and modules that provide a consistent configuration interface to remotely manage multiple RHEL systems. The **metrics** System Role configures performance analysis services for the local system and, optionally, includes a list of remote systems to be monitored by the local system. The **metrics** System Role enables you to use **pcp** to monitor your systems performance without having to configure **pcp** separately, as the set-up and deployment of **pcp** is handled by the playbook.

Table 21.1. metrics system role variables

Role variable	Description	Example usage
metrics_monitored_hosts	List of remote hosts to be analyzed by the target host. These hosts will have metrics recorded on the target host, so ensure enough disk space exists below **/var/log** for each host.	**metrics_monitored_hosts: ["*webserver.example.com*", "*database.example.com*"]**
metrics_retention_days	Configures the number of days for performance data retention before deletion.	**metrics_retention_days: 14**
metrics_graph_service	A boolean flag that enables the host to be set up with services for performance data visualization via **pcp** and **grafana**. Set to false by default.	**metrics_graph_service: no**
metrics_query_service	A boolean flag that enables the host to be set up with time series query services for querying recorded **pcp** metrics via **redis**. Set to false by default.	**metrics_query_service: no**
metrics_provider	Specifies which metrics collector to use to provide metrics. Currently, **pcp** is the only supported metrics provider.	**metrics_provider: "pcp"**

NOTE

For details about the parameters used in **metrics_connections** and additional information about the **metrics** System Role, see the **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** file.

21.2. USING THE METRICS SYSTEM ROLE TO MONITOR YOUR LOCAL SYSTEM WITH VISUALIZATION

This procedure describes how to use the **metrics** RHEL System Role to monitor your local system while simultaneously provisioning data visualization via **Grafana**.

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the **rhel-system-roles** package installed on the machine you want to monitor.

Procedure

1. Configure **localhost** in the **/etc/ansible/hosts** Ansible inventory by adding the following content to the inventory:

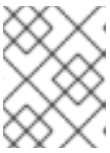
```
localhost ansible_connection=local
```

2. Create an Ansible playbook with the following content:

```
---
- hosts: localhost
  vars:
    metrics_graph_service: yes
  roles:
    - rhel-system-roles.metrics
```

3. Run the Ansible playbook:

```
# ansible-playbook name_of_your_playbook.yml
```



NOTE

Since the **metrics_graph_service** boolean is set to value="yes", **Grafana** is automatically installed and provisioned with **pcp** added as a data source.

4. To view visualization of the metrics being collected on your machine, access the **grafana** web interface as described in Accessing the Grafana web UI.

21.3. USING THE METRICS SYSTEM ROLE TO SETUP A FLEET OF INDIVIDUAL SYSTEMS TO MONITOR THEMSELVES

This procedure describes how to use the **metrics** System Role to set up a fleet of machines to monitor themselves.

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the **rhel-system-roles** package installed on the machine you want to use to run the playbook.
- You have the SSH connection established.

Procedure

1. Add the name or IP of the machines you wish to monitor via the playbook to the **/etc/ansible/hosts** Ansible inventory file under an identifying group name enclosed in brackets:

```
[remotes]
webserver.example.com
database.example.com
```

2. Create an Ansible playbook with the following content:

```
---
- hosts: remotes
  vars:
    metrics_retention_days: 0
  roles:
    - rhel-system-roles.metrics
```

3. Run the Ansible playbook:

```
# ansible-playbook name_of_your_playbook.yml -k
```

Where the **-k** prompt for password to connect to remote system.

21.4. USING THE METRICS SYSTEM ROLE TO MONITOR A FLEET OF MACHINES CENTRALLY VIA YOUR LOCAL MACHINE

This procedure describes how to use the **metrics** System Role to set up your local machine to centrally monitor a fleet of machines while also provisioning visualization of the data via **grafana** and querying of the data via **redis**.

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the **rhel-system-roles** package installed on the machine you want to use to run the playbook.

Procedure

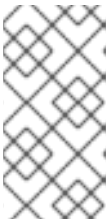
1. Create an Ansible playbook with the following content:

```
---
- hosts: localhost
```

```
vars:
  metrics_graph_service: yes
  metrics_query_service: yes
  metrics_retention_days: 10
  metrics_monitored_hosts: ["database.example.com", "webserver.example.com"]
roles:
  - rhel-system-roles.metrics
```

2. Run the Ansible playbook:

```
# ansible-playbook name_of_your_playbook.yml
```



NOTE

Since the **metrics_graph_service** and **metrics_query_service** booleans are set to value="yes", **grafana** is automatically installed and provisioned with **pcp** added as a data source with the **pcp** data recording indexed into **redis**, allowing the **pcp** querying language to be used for complex querying of the data.

3. To view graphical representation of the metrics being collected centrally by your machine and to query the data, access the **grafana** web interface as described in Accessing the Grafana web UI.

21.5. SETTING UP AUTHENTICATION WHILE MONITORING A SYSTEM USING THE METRICS SYSTEM ROLE

PCP supports the **scram-sha-256** authentication mechanism through the Simple Authentication Security Layer (SASL) framework. The **metrics** RHEL System Role automates the steps to setup authentication using the **scram-sha-256** authentication mechanism. This procedure describes how to setup authentication using the **metrics** RHEL System Role.

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the **rhel-system-roles** package installed on the machine you want to use to run the playbook.

Procedure

1. Include the following variables in the Ansible playbook you want to setup authentication for:

```
---
vars:
  metrics_username: your_username
  metrics_password: your_password
```

2. Run the Ansible playbook:

```
# ansible-playbook name_of_your_playbook.yml
```

Verification steps

- Verify the **sasl** configuration:

```
# pminfo -f -h "pcp://ip_adress?username=your_username" disk.dev.read
Password:
disk.dev.read
inst [0 or "sda"] value 19540
```

ip_adress should be replaced by the IP address of the host.

21.6. USING THE METRICS SYSTEM ROLE TO CONFIGURE AND ENABLE METRICS COLLECTION FOR SQL SERVER

This procedure describes how to use the **metrics** RHEL System Role to automate the configuration and enabling of metrics collection for Microsoft SQL Server via **pcp** on your local system.

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the **rhel-system-roles** package installed on the machine you want to monitor.
- You have installed Microsoft SQL Server for Red Hat Enterprise Linux and established a 'trusted' connection to an SQL server. See Install SQL Server and create a database on Red Hat .
- You have installed the Microsoft ODBC driver for SQL Server for Red Hat Enterprise Linux. See Red Hat Enterprise Server and Oracle Linux .

Procedure

1. Configure **localhost** in the **/etc/ansible/hosts** Ansible inventory by adding the following content to the inventory:

```
localhost ansible_connection=local
```

2. Create an Ansible playbook that contains the following content:

```
---
- hosts: localhost
  roles:
    - role: rhel-system-roles.metrics
      vars:
        metrics_from_mssql: yes
```

3. Run the Ansible playbook:

```
# ansible-playbook name_of_your_playbook.yml
```

Verification steps

- Use the **pcp** command to verify that SQL Server PMDA agent (mssql) is loaded and running:

```
# pcp
platform: Linux rhel82-2.local 4.18.0-167.el8.x86_64 #1 SMP Sun Dec 15 01:24:23 UTC
```

```
2019 x86_64
 hardware: 2 cpus, 1 disk, 1 node, 2770MB RAM
 timezone: PDT+7
 services: pmcd pmproxy
    pmcd: Version 5.0.2-1, 12 agents, 4 clients
    pmda: root pmcd proc pmproxy xfs linux nfsclient mmv kvm mssql
          jbd2 dm
 pmlogger: primary logger: /var/log/pcp/pmlogger/rhel82-2.local/20200326.16.31
    pmie: primary engine: /var/log/pcp/pmie/rhel82-2.local/pmie.log
```

Additional resources

- For more information about using Performance Co-Pilot for Microsoft SQL Server, see this Red Hat Developers Blog post.

CHAPTER 22. CONFIGURING MICROSOFT SQL SERVER USING THE MICROSOFT.SQL.SERVER ANSIBLE ROLE

As an administrator, you can use the **microsoft.sql.server** Ansible role to install, configure, and start Microsoft SQL Server (SQL Server). The **microsoft.sql.server** Ansible role optimizes your operating system to improve performance and throughput for the SQL Server. The role simplifies and automates the configuration of your RHEL host with recommended settings to run the SQL Server workloads.

22.1. PREREQUISITES

- 2 GB of RAM
- **root** access to the managed node where you want to configure SQL Server
- Pre-configured firewall
  You can set the **mssql_manage_firewall** variable to **true** so that the role can manage firewall automatically.

  Alternatively, enable the connection on the SQL Server TCP port set with the **mssql_tcp_port** variable. If you do not define this variable, the role defaults to the TCP port number **1443**.

  To add a new port, use:

  ```
  # firewall-cmd --add-port=xxxx/tcp --permanent
  # firewall-cmd --reload
  ```

  Replace *xxxx* with the TCP port number then reload the firewall rules.
- *Optional*: Create a file with the **.sql** extension containing the SQL statements and procedures to input them to SQL Server.

22.2. INSTALLING THE MICROSOFT.SQL.SERVER ANSIBLE ROLE

The **microsoft.sql.server** Ansible role is part of the **ansible-collection-microsoft-sql** package.

Prerequisites

- **root** access

Procedure

1. Install Ansible Core which is available in the RHEL 8 AppStream repository:

   ```
   # *yum install ansible-core*
   ```

2. Install the **microsoft.sql.server** Ansible role:

   ```
   # *yum install ansible-collection-microsoft-sql*
   ```

22.3. INSTALLING AND CONFIGURING SQL SERVER USING THE MICROSOFT.SQL.SERVER ANSIBLE ROLE

You can use the **microsoft.sql.server** Ansible role to install and configure SQL server.

Prerequisites

- The Ansible inventory is created

Procedure

1. Create a file with the **.yml** extension. For example, **mssql-server.yml**.
2. Add the following content to your **.yml** file:

```
---
- hosts: all
  vars:
    mssql_accept_microsoft_odbc_driver_17_for_sql_server_eula: true
    mssql_accept_microsoft_cli_utilities_for_sql_server_eula: true
    mssql_accept_microsoft_sql_server_standard_eula: true
    mssql_password: <password>
    mssql_edition: Developer
    mssql_tcp_port: 1443
  roles:
    - microsoft.sql.server
```

   Replace *<password>* with your SQL Server password.

3. Run the **mssql-server.yml** ansible playbook:

```
# *ansible-playbook mssql-server.yml*
```

22.4. TLS VARIABLES

You can use the following variables to configure the Transport Level Security (TLS) protocol.

Table 22.1. TLS role variables

Role variable	Description

Role variable	Description
mssql_tls_enable	This variable enables or disables TLS encryption. The **microsoft.sql.server** Ansible role performs following tasks when the variable is set to **true**: ● Copies TLS certificate to **/etc/pki/tls/certs/** on the SQL Server ● Copies private key to **/etc/pki/tls/private/** on the SQL Server ● Configures SQL Server to use TLS certificate and private key to encrypt connections When set to **false**, the TLS encryption is disabled. The role does not remove the existing certificate and private key files.
mssql_tls_cert	To define this variable, enter the path to the TLS certificate file.
mssql_tls_private_key	To define this variable, enter the path to the private key file.
mssql_tls_remote_src	Defines if the role searches for **mssql_tls_cert** and **mssql_tls_private_key** files remotely or on the control node. When set to the default **false**, the role searches for **mssql_tls_cert** or **mssql_tls_private_key** files on the Ansible control node. When set to **true**, the role searches for **mssql_tls_cert** or **mssql_tls_private_key** files on the Ansible managed node.
mssql_tls_version	Define this variable to select which TSL version to use. The default is **1.2**
mssql_tls_force	Set this variable to **true** to replace the certificate and private key files on the host. The files must exist under **/etc/pki/tls/certs/** and **/etc/pki/tls/private/** directories. The default is **false**.

22.5. ACCEPTING EULA FOR MLSERVICES

You must accept all the EULA for the open-source distributions of Python and R packages to install the required SQL Server Machine Learning Services (MLServices).

See **/usr/share/doc/mssql-server** for the license terms.

Table 22.2. SQL Server Machine Learning Services EULA variables

Role variable	Description
mssql_accept_microsoft_sql_server_standard_eula	This variable determines whether to accept the terms and conditions for installing the **mssql-conf** package. To accept the terms and conditions set this variable to **true**. The default is **false**.

22.6. ACCEPTING EULAS FOR MICROSOFT ODBC 17

You must accept all the EULAs to install the Microsoft Open Database Connectivity (ODBC) driver.

See **/usr/share/doc/msodbcsql17/LICENSE.txt** and **/usr/share/doc/mssql-tools/LICENSE.txt** for the license terms.

Table 22.3. Microsoft ODBC 17 EULA variables

Role variable	Description
mssql_accept_microsoft_odbc_driver_17_for_sql_server_eula	This variable determines whether to accept the terms and conditions for installing the **msodbcsql17** package. To accept the terms and conditions set this variable to **true**. The default is **false**.
mssql_accept_microsoft_cli_utilities_for_sql_server_eula	This variable determines whether to accept the terms and conditions for installing the **mssql-tools** package. To accept the terms and conditions set this variable to **true**. The default is **false**.

22.7. HIGH AVAILABILITY VARIABLES

You can configure high availability for Microsoft SQL Server with the variables from the table below.

Table 22.4. High availability configuration variables

Variable	Description
mssql_ha_configure	The default value is **false**. When it is set to **true**, performs the following actions: ● Configures firewall by opening a port from the **mssql_ha_listener_port** variable and enables the **high-availability** service in firewall. ● Configures SQL Server for high availability. ○ Enables Always On Health events. ○ Creates certificate on the primary replica and distributes it to other replicas. ○ Configures endpoint and availability group. ○ Configures the user from the **mssql_ha_login** variable for Pacemaker. ● Optional: Includes the System Roles **ha_cluster** role to configure Pacemaker. You must set **mssql_ha_cluster_run_role** to **true** and provide all variables that the **ha_cluster** role requires for a Pacemaker cluster configuration.
mssql_ha_replica_type	This variable specifies which type of replica you can configure on the host. You can set this variable to **primary**, **synchronous**, and **witness**. You must set it to **primary** only on one host.
mssql_ha_listener_port	The default port is **5022**. The role uses this TCP port to replicate data for an Always On availability group.
mssql_ha_cert_name	You must define the name of the certificate to secure transactions between members of an Always On availability group.
mssql_ha_master_key_password	You must set the password for the master key to use with the certificate.
mssql_ha_private_key_password	You must set the password for the private key to use with the certificate.

Variable	Description
mssql_ha_reset_cert	The default value is **false**. If it is set to **true**, resets the certificate which an Always On availability group uses.
mssql_ha_endpoint_name	You must define the name of the endpoint to configure.
mssql_ha_ag_name	You must define the name of the availability group to configure.
mssql_ha_db_names	You can define a list of the databases to replicate, otherwise the role creates a cluster without replicating databases.
mssql_ha_login	The SQL Server Pacemaker resource agent utilizes this user to perform database health checks and manage state transitions from replica to primary server.
mssql_ha_login_password	The password for the **mssql_ha_login** user in SQL Server.
mssql_ha_cluster_run_role	The default value is **false**. This variable defines if this role runs the **ha_cluster** role. Note that the **ha_cluster** role replaces the configuration of the HA cluster on specified nodes, any variables currently configured for the HA cluster are erased and overwritten. To work around this limitation, the **microsoft.sql.server** role does not set any variables for the **ha_cluster** role to ensure that it does not overwrite any existing Pacemaker configuration. If you want the **microsoft.sql.server** to run the **ha_cluster** role, set this variable to **true** and provide variables for the **ha_cluster** role with the **microsoft.sql.server** role call.

Note, this role backs up the database to the **/var/opt/mssql/data/** directory.

For examples on how to use high availability variables for Microsoft SQL Server:

- If you install the role from Automation Hub, see the **~/.ansible/collections/ansible_collections/microsoft/sql/roles/server/README.md** file on your server.

- If you install the role from a package, open the **/usr/share/microsoft/sql-server/README.html** file in your browser.

CHAPTER 23. CONFIGURING A SYSTEM FOR SESSION RECORDING USING THE TLOG RHEL SYSTEM ROLE

With the **tlog** RHEL System Role, you can configure a system for terminal session recording on RHEL using Red Hat Ansible Automation Platform.

23.1. THE TLOG SYSTEM ROLE

You can configure a RHEL system for terminal session recording on RHEL using the **tlog** RHEL System Role.

You can configure the recording to take place per user or user group by means of the **SSSD** service.

Additional resources

- For more details on session recording in RHEL, see Recording Sessions.

23.2. COMPONENTS AND PARAMETERS OF THE TLOG SYSTEM ROLE

The Session Recording solution has the following components:

- The **tlog** utility
- System Security Services Daemon (SSSD)
- Optional: The web console interface

The parameters used for the **tlog** RHEL System Role are:

Role Variable	Description
tlog_use_sssd (default: yes)	Configure session recording with SSSD, the preferred way of managing recorded users or groups
tlog_scope_sssd (default: none)	Configure SSSD recording scope - all / some / none
tlog_users_sssd (default: [])	YAML list of users to be recorded
tlog_groups_sssd (default: [])	YAML list of groups to be recorded

- For details about the parameters used in **tlog** and additional information about the **tlog** System Role, see the **/usr/share/ansible/roles/rhel-system-roles.tlog/README.md** file.

23.3. DEPLOYING THE TLOG RHEL SYSTEM ROLE

Follow these steps to prepare and apply an Ansible playbook to configure a RHEL system to log session recording data to the systemd journal.

Prerequisites

- You have set SSH keys for access from the control node to the target system where the **tlog** System Role will be configured.
- You have at least one system that you want to configure the **tlog** System Role.
- The Ansible Core package is installed on the control machine.
- The **rhel-system-roles** package is installed on the control machine.

Procedure

1. Create a new ***playbook.yml*** file with the following content:

```
---
- name: Deploy session recording
  hosts: all
  vars:
    tlog_scope_sssd: some
    tlog_users_sssd:
      - recorded-user

  roles:
    - rhel-system-roles.tlog
```

   Where,

   - **tlog_scope_sssd**:
     - **some** specifies you want to record only certain users and groups, not **all** or **none**.
   - **tlog_users_sssd**:
     - **recorded-user** specifies the user you want to record a session from. Note that this does not add the user for you. You must set the user by yourself.

2. Optionally, verify the playbook syntax.

```
# ansible-playbook --syntax-check playbook.yml
```

3. Run the playbook on your inventory file:

```
# ansible-playbook -i IP_Address /path/to/file/playbook.yml -v
```

As a result, the playbook installs the **tlog** RHEL System Role on the system you specified. The role includes **tlog-rec-session**, a terminal session I/O logging program, that acts as the login shell for a user. It also creates an SSSD configuration drop file that can be used by the users and groups that you define. SSSD parses and reads these users and groups, and replaces their user shell with **tlog-rec-session**. Additionally, if the **cockpit** package is installed on the system, the playbook also installs the **cockpit-session-recording** package, which is a **Cockpit** module that allows you to view and play recordings in the web console interface.

Verification steps

To verify that the SSSD configuration drop file is created in the system, perform the following steps:

1. Navigate to the folder where the SSSD configuration drop file is created:

```
# cd /etc/sssd/conf.d
```

2. Check the file content:

```
# cat /etc/sssd/conf.d/sssd-session-recording.conf
```

You can see that the file contains the parameters you set in the playbook.

23.4. DEPLOYING THE TLOG RHEL SYSTEM ROLE FOR EXCLUDING LISTS OF GROUPS OR USERS

You can use the **tlog** System Role to support the SSSD session recording configuration options **exclude_users** and **exclude_groups**. Follow these steps to prepare and apply an Ansible playbook to configure a RHEL system to exclude users or groups from having their sessions recorded and logged in the systemd journal.

Prerequisites

- You have set SSH keys for access from the control node to the target system on which you want to configure the **tlog** System Role.
- You have at least one system on which you want to configure the **tlog** System Role.
- The Ansible Core package is installed on the control machine.
- The **rhel-system-roles** package is installed on the control machine.

Procedure

1. Create a new ***playbook.yml*** file with the following content:

```
---
- name: Deploy session recording excluding users and groups
  hosts: all
  vars:
    tlog_scope_sssd: all
    tlog_exclude_users_sssd:
      - jeff
      - james
    tlog_exclude_groups_sssd:
      - admins

  roles:
    - rhel-system-roles.tlog
```

Where,

- **tlog_scope_sssd**:
  - **all**: specifies that you want to record all users and groups.
- **tlog_exclude_users_sssd**:

    - user names: specifies the user names of the users you want to exclude from the session recording.

- **tlog_exclude_groups_sssd**:
    - **admins** specifies the group you want to exclude from the session recording.

2. Optionally, verify the playbook syntax;

```
# ansible-playbook --syntax-check playbook.yml
```

3. Run the playbook on your inventory file:

```
# ansible-playbook -i IP_Address /path/to/file/playbook.yml -v
```

As a result, the playbook installs the **tlog** RHEL System Role on the system you specified. The role includes **tlog-rec-session**, a terminal session I/O logging program, that acts as the login shell for a user. It also creates an **/etc/sssd/conf.d/sssd-session-recording.conf** SSSD configuration drop file that can be used by users and groups except those that you defined as excluded. SSSD parses and reads these users and groups, and replaces their user shell with **tlog-rec-session**. Additionally, if the **cockpit** package is installed on the system, the playbook also installs the **cockpit-session-recording** package, which is a **Cockpit** module that allows you to view and play recordings in the web console interface.

Verification steps

To verify that the SSSD configuration drop file is created in the system, perform the following steps:

1. Navigate to the folder where the SSSD configuration drop file is created:

```
# cd /etc/sssd/conf.d
```

2. Check the file content:

```
# cat sssd-session-recording.conf
```

You can see that the file contains the parameters you set in the playbook.

Additional resources

- See the **/usr/share/doc/rhel-system-roles/tlog/** and **/usr/share/ansible/roles/rhel-system-roles.tlog/** directories.
- The Recording a session using the deployed Terminal Session Recording System Role in the CLI .

23.5. RECORDING A SESSION USING THE DEPLOYED TLOG SYSTEM ROLE IN THE CLI

After you have deployed the **tlog** System Role in the system you have specified, you are able to record a user terminal session using the command-line interface (CLI).

Prerequisites

- You have deployed the **tlog** System Role in the target system.

- The SSSD configuration drop file was created in the **/etc/sssd/conf.d** directory. See Deploying the Terminal Session Recording RHEL System Role.

Procedure

1. Create a user and assign a password for this user:

```
# useradd recorded-user
# passwd recorded-user
```

2. Log in to the system as the user you just created:

```
# ssh recorded-user@localhost
```

3. Type "yes" when the system prompts you to type yes or no to authenticate.
4. Insert the *recorded-user's* password.
   The system displays a message about your session being recorded.

```
ATTENTION! Your session is being recorded!
```

5. After you have finished recording the session, type:

```
# exit
```

   The system logs out from the user and closes the connection with the localhost.

As a result, the user session is recorded, stored and you can play it using a journal.

Verification steps

To view your recorded session in the journal, do the following steps:

1. Run the command below:

```
# journalctl -o verbose -r
```

2. Search for the **MESSAGE** field of the **tlog-rec** recorded journal entry.

```
# journalctl -xel _EXE=/usr/bin/tlog-rec-session
```

23.6. WATCHING A RECORDED SESSION USING THE CLI

You can play a user session recording from a journal using the command-line interface (CLI).

Prerequisites

- You have recorded a user session. See Recording a session using the deployed tlog System Role in the CLI .

Procedure

1. On the CLI terminal, play the user session recording:

```
# journalctl -o verbose -r
```

2. Search for the **tlog** recording:

```
$ /tlog-rec
```

You can see details such as:

- The username for the user session recording
- The **out_txt** field, a raw output encode of the recorded session
- The identifier number TLOG_REC=*ID_number*

3. Copy the identifier number TLOG_REC=*ID_number*.
4. Playback the recording using the identifier number TLOG_REC=*ID_number*.

```
# tlog-play -r journal -M TLOG_REC=ID_number
```

As a result, you can see the user session recording terminal output being played back.

CHAPTER 24. CONFIGURING A HIGH-AVAILABILITY CLUSTER USING SYSTEM ROLES

With the **ha_cluster** System Role, you can configure and manage a high-availability cluster that uses the Pacemaker high availability cluster resource manager.

24.1. HA_CLUSTER SYSTEM ROLE VARIABLES

In an **ha_cluster** System Role playbook, you define the variables for a high availability cluster according to the requirements of your cluster deployment.

The variables you can set for an **ha_cluster** System Role are as follows.

ha_cluster_enable_repos

A boolean flag that enables the repositories containing the packages that are needed by the **ha_cluster** System Role. When this is set to **yes**, the default value of this variable, you must have active subscription coverage for RHEL and the RHEL High Availability Add-On on the systems that you will use as your cluster members or the system role will fail.

ha_cluster_cluster_present

A boolean flag which, if set to **yes**, determines that HA cluster will be configured on the hosts according to the variables passed to the role. Any cluster configuration not specified in the role and not supported by the role will be lost.
If **ha_cluster_cluster_present** is set to **no**, all HA cluster configuration will be removed from the target hosts.

The default value of this variable is **yes**.

The following example playbook removes all cluster configuration on **node1** and **node2**

```
- hosts: node1 node2
  vars:
    ha_cluster_cluster_present: no

  roles:
    - rhel-system-roles.ha_cluster
```

ha_cluster_start_on_boot

A boolean flag that determines whether cluster services will be configured to start on boot. The default value of this variable is **yes**.

ha_cluster_fence_agent_packages

List of fence agent packages to install. The default value of this variable is **fence-agents-all**, **fence-virt**.

ha_cluster_extra_packages

List of additional packages to be installed. The default value of this variable is no packages.
This variable can be used to install additional packages not installed automatically by the role, for example custom resource agents.

It is possible to specify fence agents as members of this list. However, **ha_cluster_fence_agent_packages** is the recommended role variable to use for specifying fence agents, so that its default value is overridden.

ha_cluster_hacluster_password

A string value that specifies the password of the **hacluster** user. The **hacluster** user has full access to a cluster. It is recommended that you vault encrypt the password, as described in Encrypting content with Ansible Vault. There is no default password value, and this variable must be specified.

ha_cluster_corosync_key_src

The path to Corosync **authkey** file, which is the authentication and encryption key for Corosync communication. It is highly recommended that you have a unique **authkey** value for each cluster. The key should be 256 bytes of random data.
If you specify a key for this variable, it is recommended that you vault encrypt the key, as described in Encrypting content with Ansible Vault .

If no key is specified, a key already present on the nodes will be used. If nodes do not have the same key, a key from one node will be distributed to other nodes so that all nodes have the same key. If no node has a key, a new key will be generated and distributed to the nodes.

If this variable is set, **ha_cluster_regenerate_keys** is ignored for this key.

The default value of this variable is null.

ha_cluster_pacemaker_key_src

The path to the Pacemaker **authkey** file, which is the authentication and encryption key for Pacemaker communication. It is highly recommended that you have a unique **authkey** value for each cluster. The key should be 256 bytes of random data.
If you specify a key for this variable, it is recommended that you vault encrypt the key, as described in Encrypting content with Ansible Vault .

If no key is specified, a key already present on the nodes will be used. If nodes do not have the same key, a key from one node will be distributed to other nodes so that all nodes have the same key. If no node has a key, a new key will be generated and distributed to the nodes.

If this variable is set, **ha_cluster_regenerate_keys** is ignored for this key.

The default value of this variable is null.

ha_cluster_fence_virt_key_src

The path to the **fence-virt** or **fence-xvm** pre-shared key file, which is the location of the authentication key for the **fence-virt** or **fence-xvm** fence agent.
If you specify a key for this variable, it is recommended that you vault encrypt the key, as described in Encrypting content with Ansible Vault .

If no key is specified, a key already present on the nodes will be used. If nodes do not have the same key, a key from one node will be distributed to other nodes so that all nodes have the same key. If no node has a key, a new key will be generated and distributed to the nodes. If the **ha_cluster** System Role generates a new key in this fashion, you should copy the key to your nodes' hypervisor to ensure that fencing works.

If this variable is set, **ha_cluster_regenerate_keys** is ignored for this key.

The default value of this variable is null.

ha_cluster_pcsd_public_key_srcr, ha_cluster_pcsd_private_key_src

The path to the **pcsd** TLS certificate and private key. If this is not specified, a certificate-key pair already present on the nodes will be used. If a certificate-key pair is not present, a random new one will be generated.

If you specify a private key value for this variable, it is recommended that you vault encrypt the key, as described in Encrypting content with Ansible Vault .

If these variables are set, **ha_cluster_regenerate_keys** is ignored for this certificate-key pair.

The default value of these variables is null.

ha_cluster_regenerate_keys

A boolean flag which, when set to **yes**, determines that pre-shared keys and TLS certificates will be regenerated. For more information on when keys and certificates will be regenerated, see the descriptions of the **ha_cluster_corosync_key_src**, **ha_cluster_pacemaker_key_src**, **ha_cluster_fence_virt_key_src**, **ha_cluster_pcsd_public_key_src**, and **ha_cluster_pcsd_private_key_src** variables.
The default value of this variable is **no**.

ha_cluster_pcs_permission_list

Configures permissions to manage a cluster using **pcsd**. The items you configure with this variable are as follows:

- **type** - **user** or **group**
- **name** - user or group name
- **allow_list** - Allowed actions for the specified user or group:
  - **read** - View cluster status and settings
  - **write** - Modify cluster settings except permissions and ACLs
  - **grant** - Modify cluster permissions and ACLs
  - **full** - Unrestricted access to a cluster including adding and removing nodes and access to keys and certificates

The structure of the **ha_cluster_pcs_permission_list** variable and its default values are as follows:

```
ha_cluster_pcs_permission_list:
  - type: group
    name: hacluster
    allow_list:
      - grant
      - read
      - write
```

ha_cluster_cluster_name

The name of the cluster. This is a string value with a default of **my-cluster**.

ha_cluster_transport

(RHEL 8.7 and later) Sets the cluster transport method. The items you configure with this variable are as follows:

- **type** (optional) - Transport type: **knet**, **udp**, or **udpu**. The **udp** and **udpu** transport types support only one link. Encryption is always disabled for **udp** and **udpu**. Defaults to **knet** if not specified.

- **options** (optional) - List of name-value dictionaries with transport options.
- **links** (optional) - List of list of name-value dictionaries. Each list of name-value dictionaries holds options for one Corosync link. It is recommended that you set the **linknumber** value for each link. Otherwise, the first list of dictionaries is assigned by default to the first link, the second one to the second link, and so on.
- **compression** (optional) - List of name-value dictionaries configuring transport compression. Supported only with the **knet** transport type.
- **crypto** (optional) - List of name-value dictionaries configuring transport encryption. By default, encryption is enabled. Supported only with the **knet** transport type.
  For a list of allowed options, see the **pcs -h cluster setup** help page or the **setup** description in the **cluster** section of the **pcs**(8) man page. For more detailed descriptions, see the **corosync.conf**(5) man page.

  The structure of the **ha_cluster_transport** variable is as follows:

```
ha_cluster_transport:
  type: knet
  options:
    - name: option1_name
      value: option1_value
    - name: option2_name
      value: option2_value
  links:
    -
      - name: option1_name
        value: option1_value
      - name: option2_name
        value: option2_value
    -
      - name: option1_name
        value: option1_value
      - name: option2_name
        value: option2_value
  compression:
    - name: option1_name
      value: option1_value
    - name: option2_name
      value: option2_value
  crypto:
    - name: option1_name
      value: option1_value
    - name: option2_name
      value: option2_value
```

  For an example **ha_cluster** System Role playbook that configures a transport method, see Configuring Corosync values in a high availability cluster .

ha_cluster_totem

(RHEL 8.7 and later) Configures Corosync totem. For a list of allowed options, see the **pcs -h cluster setup** help page or the **setup** description in the **cluster** section of the **pcs**(8) man page. For a more detailed description, see the **corosync.conf**(5) man page.
The structure of the **ha_cluster_totem** variable is as follows:

```
ha_cluster_totem:
  options:
    - name: option1_name
      value: option1_value
    - name: option2_name
      value: option2_value
```

For an example **ha_cluster** System Role playbook that configures a Corosync totem, see Configuring Corosync values in a high availability cluster .

ha_cluster_quorum

(RHEL 8.7 and later) Configures cluster quorum. You can configure the **auto_tie_breaker**, **last_man_standing**, **last_man_standing_window**, and **wait_for_all** quorum options. For information on quorum options, see the **votequorum**(5) man page.
The structure of the **ha_cluster_quorum** variable is as follows:

```
ha_cluster_quorum:
  options:
    - name: option1_name
      value: option1_value
    - name: option2_name
      value: option2_value
```

For an example **ha_cluster** System Role playbook that configures cluster quorum, see Configuring Corosync values in a high availability cluster.

ha_cluster_sbd_enabled

(RHEL 8.7 and later) A boolean flag which determines whether the cluster can use the SBD node fencing mechanism. The default value of this variable is **no**.
For an example **ha_cluster** System Role playbook that enables SBD, see Configuring a high availability cluster with SBD node fencing.

ha_cluster_sbd_options

(RHEL 8.7 and later) List of name-value dictionaries specifying SBD options. Supported options are:

- **delay-start** - defaults to **no**
- **startmode** - defaults to **always**
- **timeout-action** - defaults to **flush,reboot**
- **watchdog-timeout** - defaults to **5**
  For information on these options, see the **Configuration via environment** section of the **sbd**(8) man page.

  For an example **ha_cluster** System Role playbook that configures SBD options, see Configuring a high availability cluster with SBD node fencing .

  When using SBD, you can optionally configure watchdog and SBD devices for each node in an inventory. For information on configuring watchdog and SBD devices in an inventory file, see Specifying an inventory for the ha_cluster System Role .

ha_cluster_cluster_properties

List of sets of cluster properties for Pacemaker cluster-wide configuration. Only one set of cluster properties is supported.
The structure of a set of cluster properties is as follows:

```
ha_cluster_cluster_properties:
  - attrs:
      - name: property1_name
        value: property1_value
      - name: property2_name
        value: property2_value
```

By default, no properties are set.

The following example playbook configures a cluster consisting of **node1** and **node2** and sets the **stonith-enabled** and **no-quorum-policy** cluster properties.

```
- hosts: node1 node2
  vars:
    ha_cluster_cluster_name: my-new-cluster
    ha_cluster_hacluster_password: password
    ha_cluster_cluster_properties:
      - attrs:
          - name: stonith-enabled
            value: 'true'
          - name: no-quorum-policy
            value: stop

  roles:
    - rhel-system-roles.ha_cluster
```

ha_cluster_resource_primitives

This variable defines pacemaker resources configured by the System Role, including stonith resources, including stonith resources. The items you can configure for each resource are as follows:

- **id** (mandatory) - ID of a resource.
- **agent** (mandatory) - Name of a resource or stonith agent, for example **ocf:pacemaker:Dummy** or **stonith:fence_xvm**. It is mandatory to specify **stonith:** for stonith agents. For resource agents, it is possible to use a short name, such as **Dummy**, instead of **ocf:pacemaker:Dummy**. However, if several agents with the same short name are installed, the role will fail as it will be unable to decide which agent should be used. Therefore, it is recommended that you use full names when specifying a resource agent.
- **instance_attrs** (optional) - List of sets of the resource's instance attributes. Currently, only one set is supported. The exact names and values of attributes, as well as whether they are mandatory or not, depend on the resource or stonith agent.
- **meta_attrs** (optional) - List of sets of the resource's meta attributes. Currently, only one set is supported.
- **operations** (optional) - List of the resource's operations.
  - **action** (mandatory) - Operation action as defined by pacemaker and the resource or stonith agent.

- **attrs** (mandatory) - Operation options, at least one option must be specified.

The structure of the resource definition that you configure with the **ha_cluster** System Role is as follows.

```
- id: resource-id
  agent: resource-agent
  instance_attrs:
    - attrs:
        - name: attribute1_name
          value: attribute1_value
        - name: attribute2_name
          value: attribute2_value
  meta_attrs:
    - attrs:
        - name: meta_attribute1_name
          value: meta_attribute1_value
        - name: meta_attribute2_name
          value: meta_attribute2_value
  operations:
    - action: operation1-action
      attrs:
        - name: operation1_attribute1_name
          value: operation1_attribute1_value
        - name: operation1_attribute2_name
          value: operation1_attribute2_value
    - action: operation2-action
      attrs:
        - name: operation2_attribute1_name
          value: operation2_attribute1_value
        - name: operation2_attribute2_name
          value: operation2_attribute2_value
```

By default, no resources are defined.

For an example **ha_cluster** System Role playbook that includes resource configuration, see Configuring a high availability cluster with fencing and resources .

ha_cluster_resource_groups

This variable defines pacemaker resource groups configured by the System Role. The items you can configure for each resource group are as follows:

- **id** (mandatory) - ID of a group.
- **resources** (mandatory) - List of the group's resources. Each resource is referenced by its ID and the resources must be defined in the **ha_cluster_resource_primitives** variable. At least one resource must be listed.
- **meta_attrs** (optional) - List of sets of the group's meta attributes. Currently, only one set is supported.

The structure of the resource group definition that you configure with the **ha_cluster** System Role is as follows.

```
ha_cluster_resource_groups:
```

```
- id: group-id
  resource_ids:
    - resource1-id
    - resource2-id
  meta_attrs:
    - attrs:
        - name: group_meta_attribute1_name
          value: group_meta_attribute1_value
        - name: group_meta_attribute2_name
          value: group_meta_attribute2_value
```

By default, no resource groups are defined.

For an example **ha_cluster** System Role playbook that includes resource group configuration, see Configuring a high availability cluster with fencing and resources .

ha_cluster_resource_clones

This variable defines pacemaker resource clones configured by the System Role. The items you can configure for a resource clone are as follows:

- **resource_id** (mandatory) - Resource to be cloned. The resource must be defined in the **ha_cluster_resource_primitives** variable or the **ha_cluster_resource_groups** variable.
- **promotable** (optional) - Indicates whether the resource clone to be created is a promotable clone, indicated as **yes** or **no**.
- **id** (optional) - Custom ID of the clone. If no ID is specified, it will be generated. A warning will be displayed if this option is not supported by the cluster.
- **meta_attrs** (optional) - List of sets of the clone's meta attributes. Currently, only one set is supported.

The structure of the resource clone definition that you configure with the **ha_cluster** System Role is as follows.

```
ha_cluster_resource_clones:
  - resource_id: resource-to-be-cloned
    promotable: yes
    id: custom-clone-id
    meta_attrs:
      - attrs:
          - name: clone_meta_attribute1_name
            value: clone_meta_attribute1_value
          - name: clone_meta_attribute2_name
            value: clone_meta_attribute2_value
```

By default, no resource clones are defined.

For an example **ha_cluster** System Role playbook that includes resource clone configuration, see Configuring a high availability cluster with fencing and resources .

ha_cluster_constraints_location

This variable defines resource location constraints. Resource location constraints indicate which nodes a resource can run on. You can specify a resources specified by a resource ID or by a pattern, which can match more than one resource. You can specify a node by a node name or by a rule.

The items you can configure for a resource location constraint are as follows:

- **resource** (mandatory) - Specification of a resource the constraint applies to.
- **node** (mandatory) - Name of a node the resource should prefer or avoid.
- **id** (optional) - ID of the constraint. If not specified, it will be autogenerated.
- **options** (optional) - List of name-value dictionaries.
  - **score** - Sets the weight of the constraint.
    - A positive **score** value means the resource prefers running on the node.
    - A negative **score** value means the resource should avoid running on the node.
    - A **score** value of **-INFINITY** means the resource must avoid running on the node.
    - If **score** is not specified, the score value defaults to **INFINITY**.

By default no resource location constraints are defined.
The structure of a resource location constraint specifying a resource ID and node name is as follows:

```
ha_cluster_constraints_location:
  - resource:
      id: resource-id
    node: node-name
    id: constraint-id
    options:
      - name: score
        value: score-value
      - name: option-name
        value: option-value
```

The items that you configure for a resource location constraint that specifies a resource pattern are the same items that you configure for a resource location constraint that specifies a resource ID, with the exception of the resource specification itself. The item that you specify for the resource specification is as follows:

- **pattern** (mandatory) - POSIX extended regular expression resource IDs are matched against.

The structure of a resource location constraint specifying a resource pattern and node name is as follows:

```
ha_cluster_constraints_location:
  - resource:
      pattern: resource-pattern
    node: node-name
    id: constraint-id
    options:
      - name: score
        value: score-value
      - name: resource-discovery
        value: resource-discovery-value
```

The items you can configure for a resource location constraint that specifies a resource ID and a rule are as follows:

- **resource** (mandatory) - Specification of a resource the constraint applies to.
  - **id** (mandatory) - Resource ID.
  - **role** (optional) - The resource role to which the constraint is limited: **Started**, **Unpromoted**, **Promoted**.
- **rule** (mandatory) - Constraint rule written using **pcs** syntax. For further information, see the **constraint location** section of the **pcs**(8) man page.
- Other items to specify have the same meaning as for a resource constraint that does not specify a rule.

The structure of a resource location constraint that specifies a resource ID and a rule is as follows:

```
ha_cluster_constraints_location:
  - resource:
      id: resource-id
      role: resource-role
    rule: rule-string
    id: constraint-id
    options:
      - name: score
        value: score-value
      - name: resource-discovery
        value: resource-discovery-value
```

The items that you configure for a resource location constraint that specifies a resource pattern and a rule are the same items that you configure for a resource location constraint that specifies a resource ID and a rule, with the exception of the resource specification itself. The item that you specify for the resource specification is as follows:

- **pattern** (mandatory) - POSIX extended regular expression resource IDs are matched against.

The structure of a resource location constraint that specifies a resource pattern and a rule is as follows:

```
ha_cluster_constraints_location:
  - resource:
      pattern: resource-pattern
      role: resource-role
    rule: rule-string
    id: constraint-id
    options:
      - name: score
        value: score-value
      - name: resource-discovery
        value: resource-discovery-value
```

For an example **ha_cluster** system role playbook that creates a cluster with resource constraints, see Configuring a high availability cluster with resource constraints .

ha_cluster_constraints_colocation

This variable defines resource colocation constraints. Resource colocation constraints indicate that the location of one resource depends on the location of another one. There are two types of colocation constraints: a simple colocation constraint for two resources, and a set colocation constraint for multiple resources.
The items you can configure for a simple resource colocation constraint are as follows:

- **resource_follower** (mandatory) - A resource that should be located relative to **resource_leader**.
  - **id** (mandatory) - Resource ID.
  - **role** (optional) - The resource role to which the constraint is limited: **Started**, **Unpromoted**, **Promoted**.
- **resource_leader** (mandatory) - The cluster will decide where to put this resource first and then decide where to put **resource_follower**.
  - **id** (mandatory) - Resource ID.
  - **role** (optional) - The resource role to which the constraint is limited: **Started**, **Unpromoted**, **Promoted**.
- **id** (optional) - ID of the constraint. If not specified, it will be autogenerated.
- **options** (optional) - List of name-value dictionaries.
  - **score** - Sets the weight of the constraint.
    - Positive **score** values indicate the resources should run on the same node.
    - Negative **score** values indicate the resources should run on different nodes.
    - A **score** value of **+INFINITY** indicates the resources must run on the same node.
    - A **score** value of **-INFINITY** indicates the resources must run on different nodes.
    - If **score** is not specified, the score value defaults to **INFINITY**.

By default no resource colocation constraints are defined.
The structure of a simple resource colocation constraint is as follows:

```
ha_cluster_constraints_colocation:
  - resource_follower:
      id: resource-id1
      role: resource-role1
    resource_leader:
      id: resource-id2
      role: resource-role2
    id: constraint-id
    options:
      - name: score
        value: score-value
      - name: option-name
        value: option-value
```

The items you can configure for a resource set colocation constraint are as follows:

- **resource_sets** (mandatory) - List of resource sets.
  - **resource_ids** (mandatory) - List of resources in a set.
  - **options** (optional) - List of name-value dictionaries fine-tuning how resources in the sets are treated by the constraint.
- **id** (optional) - Same values as for a simple colocation constraint.
- **options** (optional) - Same values as for a simple colocation constraint.

The structure of a resource set colocation constraint is as follows:

```
ha_cluster_constraints_colocation:
  - resource_sets:
      - resource_ids:
          - resource-id1
          - resource-id2
        options:
          - name: option-name
            value: option-value
    id: constraint-id
    options:
      - name: score
        value: score-value
      - name: option-name
        value: option-value
```

For an example **ha_cluster** system role playbook that creates a cluster with resource constraints, see Configuring a high availability cluster with resource constraints .

ha_cluster_constraints_order

This variable defines resource order constraints. Resource order constraints indicate the order in which certain resource actions should occur. There are two types of resource order constraints: a simple order constraint for two resources, and a set order constraint for multiple resources.
The items you can configure for a simple resource order constraint are as follows:

- **resource_first** (mandatory) - Resource that the **resource_then** resource depends on.
  - **id** (mandatory) - Resource ID.
  - **action** (optional) - The action that must complete before an action can be initiated for the **resource_then** resource. Allowed values: **start**, **stop**, **promote**, **demote**.
- **resource_then** (mandatory) - The dependent resource.
  - **id** (mandatory) - Resource ID.
  - **action** (optional) - The action that the resource can execute only after the action on the **resource_first** resource has completed. Allowed values: **start**, **stop**, **promote**, **demote**.
- **id** (optional) - ID of the constraint. If not specified, it will be autogenerated.
- **options** (optional) - List of name-value dictionaries.

By default no resource order constraints are defined.
The structure of a simple resource order constraint is as follows:

```
ha_cluster_constraints_order:
  - resource_first:
      id: resource-id1
      action: resource-action1
    resource_then:
      id: resource-id2
      action: resource-action2
    id: constraint-id
    options:
      - name: score
        value: score-value
      - name: option-name
        value: option-value
```

The items you can configure for a resource set order constraint are as follows:

- **resource_sets** (mandatory) - List of resource sets.
  - **resource_ids** (mandatory) - List of resources in a set.
  - **options** (optional) - List of name-value dictionaries fine-tuning how resources in the sets are treated by the constraint.
- **id** (optional) - Same values as for a simple order constraint.
- **options** (optional) - Same values as for a simple order constraint.

The structure of a resource set order constraint is as follows:

```
ha_cluster_constraints_order:
  - resource_sets:
      - resource_ids:
          - resource-id1
          - resource-id2
        options:
          - name: option-name
            value: option-value
    id: constraint-id
    options:
      - name: score
        value: score-value
      - name: option-name
        value: option-value
```

For an example **ha_cluster** system role playbook that creates a cluster with resource constraints, see Configuring a high availability cluster with resource constraints .

ha_cluster_constraints_ticket

This variable defines resource ticket constraints. Resource ticket constraints indicate the resources that depend on a certain ticket. There are two types of resource ticket constraints: a simple ticket constraint for one resource, and a ticket order constraint for multiple resources.
The items you can configure for a simple resource ticket constraint are as follows:

- **resource** (mandatory) - Specification of a resource the constraint applies to.
  - **id** (mandatory) - Resource ID.
  - **role** (optional) - The resource role to which the constraint is limited: **Started**, **Unpromoted**, **Promoted**.
- **ticket** (mandatory) - Name of a ticket the resource depends on.
- **id** (optional) - ID of the constraint. If not specified, it will be autogenerated.
- **options** (optional) - List of name-value dictionaries.
  - **loss-policy** (optional) - Action to perform on the resource if the ticket is revoked.

By default no resource ticket constraints are defined.
The structure of a simple resource ticket constraint is as follows:

```
ha_cluster_constraints_ticket:
  - resource:
      id: resource-id
      role: resource-role
    ticket: ticket-name
    id: constraint-id
    options:
      - name: loss-policy
        value: loss-policy-value
      - name: option-name
        value: option-value
```

The items you can configure for a resource set ticket constraint are as follows:

- **resource_sets** (mandatory) - List of resource sets.
  - **resource_ids** (mandatory) - List of resources in a set.
  - **options** (optional) - List of name-value dictionaries fine-tuning how resources in the sets are treated by the constraint.
- **ticket** (mandatory) - Same value as for a simple ticket constraint.
- **id** (optional) - Same value as for a simple ticket constraint.
- **options** (optional) - Same values as for a simple ticket constraint.

The structure of a resource set ticket constraint is as follows:

```
ha_cluster_constraints_ticket:
  - resource_sets:
      - resource_ids:
          - resource-id1
          - resource-id2
        options:
          - name: option-name
            value: option-value
    ticket: ticket-name
```

```
    id: constraint-id
    options:
      - name: option-name
        value: option-value
```

For an example **ha_cluster** system role playbook that creates a cluster with resource constraints, see Configuring a high availability cluster with resource constraints .

24.2. SPECIFYING AN INVENTORY FOR THE HA_CLUSTER SYSTEM ROLE

When configuring an HA cluster using the **ha_cluster** System Role playbook, you configure the names and addresses of the nodes for the cluster in an inventory.

24.2.1. Configuring node names and addresses in an inventory

For each node in an inventory, you can optionally specify the following items:

- **node_name** - the name of a node in a cluster.
- **pcs_address** - an address used by **pcs** to communicate with the node. It can be a name, FQDN or an IP address and it can include a port number.
- **corosync_addresses** - list of addresses used by Corosync. All nodes which form a particular cluster must have the same number of addresses and the order of the addresses matters.

The following example shows an inventory with targets **node1** and **node2**. **node1** and **node2** must be either fully qualified domain names or must otherwise be able to connect to the nodes as when, for example, the names are resolvable through the **/etc/hosts** file.

```
all:
  hosts:
    node1:
      ha_cluster:
        node_name: node-A
        pcs_address: node1-address
        corosync_addresses:
          - 192.168.1.11
          - 192.168.2.11
    node2:
      ha_cluster:
        node_name: node-B
        pcs_address: node2-address:2224
        corosync_addresses:
          - 192.168.1.12
          - 192.168.2.12
```

24.2.2. Configuring watchdog and SBD devices in an inventory (RHEL 8.7 and later)

When using SBD, you can optionally configure watchdog and SBD devices for each node in an inventory. Even though all SBD devices must be shared to and accesible from all nodes, each node can use different names for the devices. Watchdog devices can be different for each node as well. For information on the SBD variables you can set in a system role playbook, see the entries for **ha_cluster_sbd_enabled** and **ha_cluster_sbd_options** in **ha_cluster** System Role variables.

For each node in an inventory, you can optionally specify the following items:

- **sbd_watchdog** - Watchdog device to be used by SBD. Defaults to **/dev/watchdog** if not set.
- **sbd_devices** - Devices to use for exchanging SBD messages and for monitoring. Defaults to empty list if not set.

The following example shows an inventory that configures watchdog and SBD devices for targets **node1** and **node2**.

```
all:
  hosts:
    node1:
      ha_cluster:
        sbd_watchdog: /dev/watchdog2
        sbd_devices:
          - /dev/vdx
          - /dev/vdy
    node2:
      ha_cluster:
        sbd_watchdog: /dev/watchdog1
        sbd_devices:
          - /dev/vdw
          - /dev/vdz
```

24.3. CONFIGURING A HIGH AVAILABILITY CLUSTER RUNNING NO RESOURCES

The following procedure uses the **ha_cluster** System Role, to create a high availability cluster with no fencing configured and which runs no resources.

Prerequisites

- You have **ansible-core** installed on the node from which you want to run the playbook.

NOTE

You do not need to have **ansible-core** installed on the cluster member nodes.

- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.
- The systems that you will use as your cluster members must have active subscription coverage for RHEL and the RHEL High Availability Add-On.

WARNING

The **ha_cluster** System Role replaces any existing cluster configuration on the specified nodes. Any settings not specified in the role will be lost.

Procedure

1. Create an inventory file specifying the nodes in the cluster, as described in Specifying an inventory for the **ha_cluster** System Role .
2. Create a playbook file, for example **new-cluster.yml**.

NOTE

When creating your playbook file for production, it is recommended that you vault encrypt the password, as described in Encrypting content with Ansible Vault .

The following example playbook file configures a cluster with no fencing configured and which runs no resources.

```
- hosts: node1 node2
  vars:
    ha_cluster_cluster_name: my-new-cluster
    ha_cluster_hacluster_password: password

  roles:
    - rhel-system-roles.ha_cluster
```

3. Save the file.
4. Run the playbook, specifying the path to the inventory file *inventory* you created in Step 1.

```
# ansible-playbook -i inventory new-cluster.yml
```

24.4. CONFIGURING A HIGH AVAILABILITY CLUSTER WITH FENCING AND RESOURCES

The following procedure uses the **ha_cluster** System Role to create a high availability cluster that includes a fencing device, cluster resources, resource groups, and a cloned resource.

Prerequisites

- You have **ansible-core** installed on the node from which you want to run the playbook.

NOTE

You do not need to have **ansible-core** installed on the cluster member nodes.

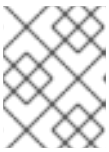
- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.
- The systems that you will use as your cluster members must have active subscription coverage for RHEL and the RHEL High Availability Add-On.

WARNING

The **ha_cluster** System Role replaces any existing cluster configuration on the specified nodes. Any settings not specified in the role will be lost.

Procedure

1. Create an inventory file specifying the nodes in the cluster, as described in Specifying an inventory for the **ha_cluster** System Role .
2. Create a playbook file, for example **new-cluster.yml**.



NOTE

When creating your playbook file for production, it is recommended that you vault encrypt the password, as described in Encrypting content with Ansible Vault .

The following example playbook file configures a cluster that includes fencing, several resources, and a resource group. It also includes a resource clone for the resource group.

```
- hosts: node1 node2
  vars:
    ha_cluster_cluster_name: my-new-cluster
    ha_cluster_hacluster_password: password
    ha_cluster_resource_primitives:
      - id: xvm-fencing
        agent: 'stonith:fence_xvm'
        instance_attrs:
          - attrs:
              - name: pcmk_host_list
                value: node1 node2
      - id: simple-resource
        agent: 'ocf:pacemaker:Dummy'
      - id: resource-with-options
        agent: 'ocf:pacemaker:Dummy'
        instance_attrs:
          - attrs:
              - name: fake
                value: fake-value
              - name: passwd
                value: passwd-value
        meta_attrs:
          - attrs:
              - name: target-role
                value: Started
              - name: is-managed
                value: 'true'
        operations:
          - action: start
            attrs:
              - name: timeout
```

```
            value: '30s'
        - action: monitor
          attrs:
            - name: timeout
              value: '5'
            - name: interval
              value: '1min'
    - id: dummy-1
      agent: 'ocf:pacemaker:Dummy'
    - id: dummy-2
      agent: 'ocf:pacemaker:Dummy'
    - id: dummy-3
      agent: 'ocf:pacemaker:Dummy'
    - id: simple-clone
      agent: 'ocf:pacemaker:Dummy'
    - id: clone-with-options
      agent: 'ocf:pacemaker:Dummy'
  ha_cluster_resource_groups:
    - id: simple-group
      resource_ids:
        - dummy-1
        - dummy-2
      meta_attrs:
        - attrs:
            - name: target-role
              value: Started
            - name: is-managed
              value: 'true'
    - id: cloned-group
      resource_ids:
        - dummy-3
  ha_cluster_resource_clones:
    - resource_id: simple-clone
    - resource_id: clone-with-options
      promotable: yes
      id: custom-clone-id
      meta_attrs:
        - attrs:
            - name: clone-max
              value: '2'
            - name: clone-node-max
              value: '1'
    - resource_id: cloned-group
      promotable: yes

roles:
  - rhel-system-roles.ha_cluster
```

3. Save the file.

4. Run the playbook, specifying the path to the inventory file *inventory* you created in Step 1.

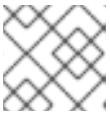
```
# ansible-playbook -i inventory new-cluster.yml
```

24.5. CONFIGURING A HIGH AVAILABILITY CLUSTER WITH RESOURCE CONSTRAINTS

The following procedure uses the **ha_cluster** system role to create a high availability cluster that includes resource location constraints, resource colocation constraints, resource order constraints, and resource ticket constraints.

Prerequisites

- You have **ansible-core** installed on the node from which you want to run the playbook.



NOTE

You do not need to have **ansible-core** installed on the cluster member nodes.

- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.
- The systems that you will use as your cluster members must have active subscription coverage for RHEL and the RHEL High Availability Add-On.

WARNING

The **ha_cluster** system role replaces any existing cluster configuration on the specified nodes. Any settings not specified in the role will be lost.

Procedure

1. Create an inventory file specifying the nodes in the cluster, as described in Specifying an inventory for the ha_cluster system role .
2. Create a playbook file, for example **new-cluster.yml**.

NOTE

When creating your playbook file for production, it is recommended that you vault encrypt the password, as described in Encrypting content with Ansible Vault .

The following example playbook file configures a cluster that includes resource location constraints, resource colocation constraints, resource order constraints, and resource ticket constraints.

```
- hosts: node1 node2
  vars:
    ha_cluster_cluster_name: my-new-cluster
    ha_cluster_hacluster_password: password
    # In order to use constraints, we need resources the constraints will apply
    # to.
    ha_cluster_resource_primitives:
```

```
  - id: xvm-fencing
    agent: 'stonith:fence_xvm'
    instance_attrs:
      - attrs:
          - name: pcmk_host_list
            value: node1 node2
  - id: dummy-1
    agent: 'ocf:pacemaker:Dummy'
  - id: dummy-2
    agent: 'ocf:pacemaker:Dummy'
  - id: dummy-3
    agent: 'ocf:pacemaker:Dummy'
  - id: dummy-4
    agent: 'ocf:pacemaker:Dummy'
  - id: dummy-5
    agent: 'ocf:pacemaker:Dummy'
  - id: dummy-6
    agent: 'ocf:pacemaker:Dummy'
# location constraints
ha_cluster_constraints_location:
  # resource ID and node name
  - resource:
      id: dummy-1
    node: node1
    options:
      - name: score
        value: 20
  # resource pattern and node name
  - resource:
      pattern: dummy-\d+
    node: node1
    options:
      - name: score
        value: 10
  # resource ID and rule
  - resource:
      id: dummy-2
    rule: '#uname eq node2 and date in_range 2022-01-01 to 2022-02-28'
  # resource pattern and rule
  - resource:
      pattern: dummy-\d+
    rule: node-type eq weekend and date-spec weekdays=6-7
# colocation constraints
ha_cluster_constraints_colocation:
  # simple constraint
  - resource_leader:
      id: dummy-3
    resource_follower:
      id: dummy-4
    options:
      - name: score
        value: -5
  # set constraint
  - resource_sets:
      - resource_ids:
          - dummy-1
```

```
          - dummy-2
      - resource_ids:
          - dummy-5
          - dummy-6
        options:
          - name: sequential
            value: "false"
    options:
      - name: score
        value: 20
# order constraints
ha_cluster_constraints_order:
  # simple constraint
  - resource_first:
      id: dummy-1
    resource_then:
      id: dummy-6
    options:
      - name: symmetrical
        value: "false"
  # set constraint
  - resource_sets:
      - resource_ids:
          - dummy-1
          - dummy-2
        options:
          - name: require-all
            value: "false"
          - name: sequential
            value: "false"
      - resource_ids:
          - dummy-3
      - resource_ids:
          - dummy-4
          - dummy-5
        options:
          - name: sequential
            value: "false"
# ticket constraints
ha_cluster_constraints_ticket:
  # simple constraint
  - resource:
      id: dummy-1
    ticket: ticket1
    options:
      - name: loss-policy
        value: stop
  # set constraint
  - resource_sets:
      - resource_ids:
          - dummy-3
          - dummy-4
          - dummy-5
    ticket: ticket2
    options:
      - name: loss-policy
```

```
            value: fence

roles:
  - linux-system-roles.ha_cluster
```

3. Save the file.
4. Run the playbook, specifying the path to the inventory file *inventory* you created in Step 1.

```
# ansible-playbook -i inventory new-cluster.yml
```

24.6. CONFIGURING COROSYNC VALUES IN A HIGH AVAILABILITY CLUSTER

(RHEL 8.7 and later) The following procedure uses the **ha_cluster** System Role to create a high availability cluster that configures Corosync values.

Prerequisites

- You have **ansible-core** installed on the node from which you want to run the playbook.

NOTE

You do not need to have **ansible-core** installed on the cluster member nodes.

- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.
  For details about RHEL System Roles and how to apply them, see Getting started with RHEL System Roles.
- The systems that you will use as your cluster members have active subscription coverage for RHEL and the RHEL High Availability Add-On.

WARNING

The **ha_cluster** System Role replaces any existing cluster configuration on the specified nodes. Any settings not specified in the role will be lost.

Procedure

1. Create an inventory file specifying the nodes in the cluster, as described in Specifying an inventory for the **ha_cluster** System Role.
2. Create a playbook file, for example **new-cluster.yml**.

NOTE

When creating your playbook file for production, it is recommended that you vault encrypt the password, as described in Encrypting content with Ansible Vault .

The following example playbook file configures a cluster that configures Corosync properties.

```
- hosts: node1 node2
  vars:
    ha_cluster_cluster_name: my-new-cluster
    ha_cluster_hacluster_password: password
    ha_cluster_transport:
      type: knet
      options:
        - name: ip_version
          value: ipv4-6
        - name: link_mode
          value: active
      links:
        -
          - name: linknumber
            value: 1
          - name: link_priority
            value: 5
        -
          - name: linknumber
            value: 0
          - name: link_priority
            value: 10
    compression:
      - name: level
        value: 5
      - name: model
        value: zlib
    crypto:
      - name: cipher
        value: none
      - name: hash
        value: none
    ha_cluster_totem:
      options:
        - name: block_unlisted_ips
          value: 'yes'
        - name: send_join
          value: 0
    ha_cluster_quorum:
      options:
        - name: auto_tie_breaker
          value: 1
        - name: wait_for_all
          value: 1

  roles:
    - linux-system-roles.ha_cluster
```

3. Save the file.

4. Run the playbook, specifying the path to the inventory file *inventory* you created in Step 1.

```
# ansible-playbook -i inventory new-cluster.yml
```

24.7. CONFIGURING A HIGH AVAILABILITY CLUSTER WITH SBD NODE FENCING

(RHEL 8.7 and later) The following procedure uses the **ha_cluster** System Role to create a high availability cluster that uses SBD node fencing.

Prerequisites

- You have **ansible-core** installed on the node from which you want to run the playbook.

NOTE

You do not need to have **ansible-core** installed on the cluster member nodes.

- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.
  For details about RHEL System Roles and how to apply them, see Getting started with RHEL System Roles.

- The systems that you will use as your cluster members must have active subscription coverage for RHEL and the RHEL High Availability Add-On.

WARNING

The **ha_cluster** System Role replaces any existing cluster configuration on the specified nodes. Any settings not specified in the role will be lost.

Procedure

1. Create an inventory file specifying the nodes in the cluster, as described in Specifying an inventory for the **ha_cluster** System Role You can optionally configure watchdog and SBD devices for each node in the cluster in an inventory file.

2. Create a playbook file, for example **new-cluster.yml**.

NOTE

When creating your playbook file for production, it is recommended that you vault encrypt the password, as described in Encrypting content with Ansible Vault .

The following example playbook file configures a cluster that uses SBD fencing.

```
- hosts: node1 node2
  vars:
    ha_cluster_cluster_name: my-new-cluster
    ha_cluster_hacluster_password: password
    ha_cluster_sbd_enabled: yes
    ha_cluster_sbd_options:
      - name: delay-start
        value: 'no'
      - name: startmode
        value: always
      - name: timeout-action
        value: 'flush,reboot'
      - name: watchdog-timeout
        value: 5

  roles:
    - linux-system-roles.ha_cluster
```

3. Save the file.
4. Run the playbook, specifying the path to the inventory file *inventory* you created in Step 1.

```
# ansible-playbook -i inventory new-cluster.yml
```

24.8. CONFIGURING AN APACHE HTTP SERVER IN A HIGH AVAILABILITY CLUSTER WITH THE HA_CLUSTER SYSTEM ROLE

This procedure configures an active/passive Apache HTTP server in a two-node Red Hat Enterprise Linux High Availability Add-On cluster using the **ha_cluster** System Role.

Prerequisites

- You have **ansible-core** installed on the node from which you want to run the playbook.

NOTE

You do not need to have **ansible-core** installed on the cluster member nodes.

- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.
- The systems that you will use as your cluster members must have active subscription coverage for RHEL and the RHEL High Availability Add-On.
- Your system includes a public virtual IP address, required for Apache.
- Your system includes shared storage for the nodes in the cluster, using iSCSI, Fibre Channel, or other shared network block device.
- You have configured an LVM logical volume with an ext4 files system, as described in Configuring an LVM volume with an ext4 file system in a Pacemaker cluster .

- You have configured an Apache HTTP server, as described in Configuring an Apache HTTP Server.
- Your system includes an APC power switch that will be used to fence the cluster nodes.

WARNING

The **ha_cluster** System Role replaces any existing cluster configuration on the specified nodes. Any settings not specified in the role will be lost.

Procedure

1. Create an inventory file specifying the nodes in the cluster, as described in Specifying an inventory for the **ha_cluster** System Role .
2. Create a playbook file, for example **http-cluster.yml**.

NOTE

When creating your playbook file for production, it is recommended that you vault encrypt the password, as described in Encrypting content with Ansible Vault .

The following example playbook file configures a previously-created Apache HTTP server in an active/passive two-node HA cluster

This example uses an APC power switch with a host name of **zapc.example.com**. If the cluster does not use any other fence agents, you can optionally list only the fence agents your cluster requires when defining the **ha_cluster_fence_agent_packages** variable, as in this example.

```
- hosts: z1.example.com z2.example.com
  roles:
    - rhel-system-roles.ha_cluster
  vars:
    ha_cluster_hacluster_password: password
    ha_cluster_cluster_name: my_cluster
    ha_cluster_fence_agent_packages:
      - fence-agents-apc-snmp
    ha_cluster_resource_primitives:
      - id: myapc
        agent: stonith:fence_apc_snmp
        instance_attrs:
          - attrs:
              - name: ipaddr
                value: zapc.example.com
              - name: pcmk_host_map
                value: z1.example.com:1;z2.example.com:2
              - name: login
                value: apc
              - name: passwd
                value: apc
```

```
    - id: my_lvm
      agent: ocf:heartbeat:LVM-activate
      instance_attrs:
        - attrs:
            - name: vgname
              value: my_vg
            - name: vg_access_mode
              value: system_id
    - id: my_fs
      agent: Filesystem
      instance_attrs:
        - attrs:
            - name: device
              value: /dev/my_vg/my_lv
            - name: directory
              value: /var/www
            - name: fstype
              value: ext4
    - id: VirtualIP
      agent: IPaddr2
      instance_attrs:
        - attrs:
            - name: ip
              value: 198.51.100.3
            - name: cidr_netmask
              value: 24
    - id: Website
      agent: apache
      instance_attrs:
        - attrs:
            - name: configfile
              value: /etc/httpd/conf/httpd.conf
            - name: statusurl
              value: http://127.0.0.1/server-status
  ha_cluster_resource_groups:
    - id: apachegroup
      resource_ids:
        - my_lvm
        - my_fs
        - VirtualIP
        - Website
```

3. Save the file.

4. Run the playbook, specifying the path to the inventory file *inventory* you created in Step 1.

```
# ansible-playbook -i inventory http-cluster.yml
```

Verification steps

1. From one of the nodes in the cluster, check the status of the cluster. Note that all four resources are running on the same node, **z1.example.com**.
   If you find that the resources you configured are not running, you can run the **pcs resource debug-start *resource*** command to test the resource configuration.

```
[root@z1 ~]# pcs status
Cluster name: my_cluster
Last updated: Wed Jul 31 16:38:51 2013
Last change: Wed Jul 31 16:42:14 2013 via crm_attribute on z1.example.com
Stack: corosync
Current DC: z2.example.com (2) - partition with quorum
Version: 1.1.10-5.el7-9abe687
2 Nodes configured
6 Resources configured

Online: [ z1.example.com z2.example.com ]

Full list of resources:
 myapc  (stonith:fence_apc_snmp):       Started z1.example.com
 Resource Group: apachegroup
     my_lvm     (ocf::heartbeat:LVM-activate):   Started z1.example.com
     my_fs      (ocf::heartbeat:Filesystem):    Started z1.example.com
     VirtualIP  (ocf::heartbeat:IPaddr2):       Started z1.example.com
     Website    (ocf::heartbeat:apache):        Started z1.example.com
```

2. Once the cluster is up and running, you can point a browser to the IP address you defined as the **IPaddr2** resource to view the sample display, consisting of the simple word "Hello".

```
Hello
```

3. To test whether the resource group running on **z1.example.com** fails over to node **z2.example.com**, put node **z1.example.com** in **standby** mode, after which the node will no longer be able to host resources.

```
[root@z1 ~]# pcs node standby z1.example.com
```

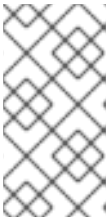
4. After putting node **z1** in **standby** mode, check the cluster status from one of the nodes in the cluster. Note that the resources should now all be running on **z2**.

```
[root@z1 ~]# pcs status
Cluster name: my_cluster
Last updated: Wed Jul 31 17:16:17 2013
Last change: Wed Jul 31 17:18:34 2013 via crm_attribute on z1.example.com
Stack: corosync
Current DC: z2.example.com (2) - partition with quorum
Version: 1.1.10-5.el7-9abe687
2 Nodes configured
6 Resources configured

Node z1.example.com (1): standby
Online: [ z2.example.com ]

Full list of resources:

 myapc  (stonith:fence_apc_snmp):       Started z1.example.com
 Resource Group: apachegroup
     my_lvm     (ocf::heartbeat:LVM-activate):   Started z2.example.com
     my_fs      (ocf::heartbeat:Filesystem):    Started z2.example.com
     VirtualIP  (ocf::heartbeat:IPaddr2):       Started z2.example.com
     Website    (ocf::heartbeat:apache):        Started z2.example.com
```

- The web site at the defined IP address should still display, without interruption.

5. To remove **z1** from **standby** mode, enter the following command.

```
[root@z1 ~]# pcs node unstandby z1.example.com
```



NOTE

Removing a node from **standby** mode does not in itself cause the resources to fail back over to that node. This will depend on the **resource-stickiness** value for the resources. For information on the **resource-stickiness** meta attribute, see Configuring a resource to prefer its current node .

24.9. ADDITIONAL RESOURCES

- Preparing a control node and managed nodes to use RHEL System Roles
- Documentation installed with the **rhel-system-roles** package in **/usr/share/ansible/roles/rhel-system-roles.logging/README.html**
- RHEL System Roles KB article
- The **ansible-playbook(1)** man page.

CHAPTER 25. INSTALLING AND CONFIGURING WEB CONSOLE WITH THE COCKPIT RHEL SYSTEM ROLE

With the **cockpit** RHEL System Role, you can install and configure the web console in your system.

25.1. THE COCKPIT SYSTEM ROLE

You can use the **cockpit** System Role to automatically deploy and enable the web console and thus be able to manage your RHEL systems from a web browser.

25.2. VARIABLES FOR THE COCKPIT RHEL SYSTEM ROLE

The parameters used for the **cockpit** RHEL System Roles are:

Role Variable	Description
cockpit_packages: (default: default)	Set one of the predefined package sets: default, minimal, or full. * cockpit_packages: (default: default) - most common pages and on-demand install UI * cockpit_packages: (default: minimal) - just the Overview, Terminal, Logs, Accounts, and Metrics pages; minimal dependencies * cockpit_packages: (default: full) - all available pages Optionally, specify your own selection of cockpit packages you want to install.
cockpit_enabled: (default:yes)	Configure if web console web server is enabled to start automatically at boot
cockpit_started: (default:yes)	Configure if web console should be started
cockpit_config: (default: nothing)	You can apply settings in the **/etc/cockpit/cockpit.conf** file. NOTE: The previous settings file will be lost.

Additional resources

- The **/usr/share/ansible/roles/rhel-system-roles.cockpit/README.md** file.
- The Cockpit configuration file man page.

25.3. INSTALLING WEB CONSOLE BY USING THE COCKPIT RHEL SYSTEM ROLE

Follow the below steps to install web console in your system and make the services accessible in it.

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **vpn** System Role.
- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.
  On the control node:
  - The **ansible-core** and **rhel-system-roles** packages are installed.
  - An inventory file which lists the managed nodes.

Procedure

1. Create a new ***playbook.yml*** file with the following content:

```
---
- hosts: all
  tasks:
    - name: Install RHEL web console
      include_role:
        name: rhel-system-roles.cockpit
      vars:
        cockpit_packages: default
        #cockpit_packages: minimal
        #cockpit_packages: full

    - name: Configure Firewall for web console
      include_role:
        name: rhel-system-roles.firewall
      vars:
        firewall:
          service: cockpit
          state: enabled
```

NOTE

The cockpit port is open by default in firewalld, so the "Configure Firewall for web console" task only applies if the system administrator customized this.

2. Optional: Verify playbook syntax.

```
# ansible-playbook --syntax-check -i inventory_file playbook.yml
```

3. Run the playbook on your inventory file:

```
# ansible-playbook -i inventory_file /path/to/file/playbook.yml
```

Additional resources

- Installing and enabling the web console .

25.4. SETTING UP A NEW CERTIFICATE BY USING THE CERTIFICATE RHEL SYSTEM ROLE

By default, web console creates a self-signed certificate on first startup. You can customize the self-signed certificate for security reasons. To generate a new certificate, you can use the **certificate** role. For that, follow the steps:

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **vpn** System Role.
- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.
  On the control node:
  - The **ansible-core** and **rhel-system-roles** packages are installed.
  - An inventory file which lists the managed nodes.

Procedure

1. Create a new ***playbook2.yml*** file with the following content:

```
---
- hosts: all
  tasks:
    - name: Generate Cockpit web server certificate
      include_role:
        name: rhel-system-roles.certificate
      vars:
        certificate_requests:
          - name: /etc/cockpit/ws-certs.d/01-certificate
            dns: ['localhost', 'www.example.com']
            ca: ipa
            group: cockpit-ws
```

2. Optional: Verify playbook syntax.

```
# ansible-playbook --syntax-check -i inventory_file playbook2.yml
```

3. Run the playbook on your inventory file:

```
# ansible-playbook -i inventory_file /path/to/file/playbook2.yml
```

Additional resources

- Requesting certificates using RHEL System Roles .