Red Hat Enterprise Linux 8

Automating system administration by using RHEL system roles

Consistent and repeatable configuration of RHEL deployments across multiple hosts with Red Hat Ansible Automation Platform playbooks

Last Updated: 2025-05-20

Red Hat Enterprise Linux 8 Automating system administration by using RHEL system roles

Consistent and repeatable configuration of RHEL deployments across multiple hosts with Red Hat Ansible Automation Platform playbooks

Abstract

The Red Hat Enterprise Linux (RHEL) system roles are a collection of Ansible roles, modules, and playbooks that help automate the consistent and repeatable administration of RHEL systems. With RHEL system roles, you can efficiently manage large inventories of systems by running configuration playbooks from a single system.

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

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

Submitting feedback through Jira (account required)

1. Log in to the Jira website.
2. Click **Create** in the top navigation bar.
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 **Create** at the bottom of the dialogue.

CHAPTER 1. INTRODUCTION TO RHEL SYSTEM ROLES

By using RHEL system roles, you can remotely manage the system configurations of multiple RHEL systems across major versions of RHEL.

Important terms and concepts

The following describes important terms and concepts in an Ansible environment:

Control node

A control node is the system from which you run Ansible commands and playbooks. Your control node can be an Ansible Automation Platform, Red Hat Satellite, or a RHEL 9, 8, or 7 host. For more information, see Preparing a control node on RHEL 8 .

Managed node

Managed nodes are the servers and network devices that you manage with Ansible. Managed nodes are also sometimes called hosts. Ansible does not have to be installed on managed nodes. For more information, see Preparing a managed node .

Ansible playbook

In a playbook, you define the configuration you want to achieve on your managed nodes or a set of steps for the system on the managed node to perform. Playbooks are Ansible's configuration, deployment, and orchestration language.

Inventory

In an inventory file, you list the managed nodes and specify information such as IP address for each managed node. In the inventory, you can also organize the managed nodes by creating and nesting groups for easier scaling. An inventory file is also sometimes called a hostfile.

Available roles and modules on a Red Hat Enterprise Linux 8 control node

Roles provided by the **rhel-system-roles** package:

- **ad_integration**: Active Directory integration
- **bootloader**: GRUB boot loader management
- **certificate**: Certificate issuance and renewal
- **cockpit**: Web console installation and configuration
- **crypto_policies**: System-wide cryptographic policies
- **fapolicy**: File access policy daemon configuration
- **firewall**: Firewalld management
- **ha_cluster**: HA Cluster management
- **journald**: Systemd journald management
- **kdump**: Kernel Dumps management
- **kernel_settings**: Kernel settings management
- **logging**: Configuring logging
- **metrics**: Performance monitoring and metrics

- **nbde_client**: Network Bound Disk Encryption client
- **nbde_server**: Network Bound Disk Encryption server
- **network**: Networking configuration
- **podman**: Podman container management
- **postfix**: Postfix configuration
- **postgresql**: PostgreSQL configuration
- **rhc**: Subscribing RHEL and configuring Insights client
- **selinux**: SELinux management
- **ssh**: SSH client configuration
- **sshd**: SSH server configuration
- **storage**: Storage management
- **systemd**: Managing systemd units
- **timesync**: Time synchronization
- **tlog**: Terminal session recording
- **vpn**: Configuring IPsec VPNs

Roles provided by the **ansible-collection-microsoft-sql** package:

- **microsoft.sql.server**: Microsoft SQL Server

Modules provided by the **ansible-collection-redhat-rhel_mgmt** package:

- **rhel_mgmt.ipmi_boot**: Setting boot devices
- **rhel_mgmt.ipmi_power**: Setting the system power state
- **rhel_mgmt.redfish_command**: Managing out-of-band controllers (OOB)
- **rhel_mgmt.redfish_command**: Querying information from OOB controllers
- **rhel_mgmt.redfish_command**: Managing BIOS, UEFI, and OOB controllers

Additional resources

- Red Hat Enterprise Linux (RHEL) system roles
- **/usr/share/ansible/roles/rhel-system-roles.*<role_name>*/README.md** file
- **/usr/share/doc/rhel-system-roles/*<role_name>*/** directory

CHAPTER 2. 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, you must prepare the control node and managed nodes.

2.1. PREPARING A CONTROL NODE ON RHEL 8

Before using RHEL system roles, you must configure a control node. This system then configures the managed hosts from the inventory according to the playbooks.

Prerequisites

- RHEL 8.6 or later is installed. For more information about installing RHEL, see Interactively installing RHEL from installation media.

> **NOTE**
>
> In RHEL 8.5 and earlier versions, Ansible packages were provided through Ansible Engine instead of Ansible Core, and with a different level of support. Do not use Ansible Engine because the packages might not be compatible with Ansible automation content in RHEL 8.6 and later. For more information, see Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories.

- The system is registered to the Customer Portal.
- A **Red Hat Enterprise Linux Server** subscription is attached to the system.
- Optional: An **Ansible Automation Platform** subscription is attached to the system.

Procedure

1. Create a user named **ansible** to manage and run playbooks:

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

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

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

   Perform the rest of the procedure as this user.

3. 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):
Enter passphrase (empty for no passphrase): <password>
Enter same passphrase again: <password>
...
```

   Use the suggested default location for the key file.

4. Optional: To prevent Ansible from prompting you for the SSH key password each time you establish a connection, configure an SSH agent.

5. 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
```

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

With these settings, Ansible performs the following actions:

- Manages hosts in the specified inventory file.
- Uses the account set in the **remote_user** parameter when it establishes SSH connections to managed nodes.
- Uses the **sudo** utility to execute tasks on managed nodes as the **root** user.
- Prompts for the root password of the remote user every time you apply a playbook. This is recommended for security reasons.

6. Create an **~/inventory** file in INI or YAML format that lists the hostnames of managed hosts. You can also define groups of hosts in 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.

7. Install RHEL system roles:

- On a RHEL host without Ansible Automation Platform, install the **rhel-system-roles** package:

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

This command installs the collections in the **/usr/share/ansible/collections/ansible_collections/redhat/rhel_system_roles/** directory, and the **ansible-core** package as a dependency.

- On Ansible Automation Platform, perform the following steps as the **ansible** user:
  i. Define Red Hat automation hub as the primary source for content in the **~/.ansible.cfg** file.
  ii. Install the **redhat.rhel_system_roles** collection from Red Hat automation hub:

```
[ansible@control-node]$ ansible-galaxy collection install redhat.rhel_system_roles
```

  This command installs the collection in the **~/.ansible/collections/ansible_collections/redhat/rhel_system_roles/** directory.

Next step

- Prepare the managed nodes. For more information, see Preparing a managed node.

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 (Red Hat Knowledgebase)
- The **ssh-keygen(1)** manual 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

2.2. PREPARING A MANAGED NODE

Managed nodes are the systems listed in the inventory and which will be configured by the control node according to the playbook. You do not have to install Ansible on managed hosts.

Prerequisites

- You prepared the control node. For more information, see Preparing a control node on RHEL 8.
- You have SSH access from the control node.

> **IMPORTANT**
>
> Direct SSH access as the **root** user is a security risk. To reduce this risk, you will create a local user on this node and configure a **sudo** policy when preparing a managed node. Ansible on the control node can then use the local user account to log in to the managed node and run playbooks as different users, such as **root**.

Procedure

1. Create a user named **ansible**:

   ```
   [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 for 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 in to 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.
   ```

   b. When prompted, connect by entering **yes**:

   ```
   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
   ```

   c. When prompted, enter the password:

   ```
   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.
   ```

d. Verify the SSH connection by remotely executing a command on the control node:

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

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

   a. Create and edit the **/etc/sudoers.d/ansible** file by using the **visudo** command:

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

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

   b. Configure a **sudoers** policy in the **/etc/sudoers.d/ansible** file that meets your requirements, for example:

   - To grant permissions to the **ansible** user to run all commands as any user and group on this host after entering the **ansible** user's password, use:

     ```
     ansible   ALL=(ALL) ALL
     ```

   - To grant permissions to the **ansible** user to run all commands as any user and group on this host without entering the **ansible** user's password, use:

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

   Alternatively, configure a more fine-granular policy that matches your security requirements. For further details on **sudoers** policies, see the **sudoers(5)** manual page.

Verification

1. Verify that you can execute commands from the control node on an all managed nodes:

   ```
   [ansible@control-node]$ ansible all -m ping
   BECOME password: <password>
   managed-node-01.example.com | SUCCESS => {
         ...
       "ping": "pong"
   }
   ...
   ```

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

2. Verify that privilege escalation works correctly by running the **whoami** utility on all managed nodes by using the Ansible **command** module:

   ```
   [ansible@control-node]$ ansible all -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.

Additional resources

- Preparing a control node on RHEL 8
- **sudoers(5)** manual page

CHAPTER 3. ANSIBLE VAULT

Sometimes your playbook needs to use sensitive data such as passwords, API keys, and other secrets to configure managed hosts. Storing this information in plain text in variables or other Ansible-compatible files is a security risk because any user with access to those files can read the sensitive data.

With Ansible vault, you can encrypt, decrypt, view, and edit sensitive information. They could be included as:

- Inserted variable files in an Ansible Playbook
- Host and group variables
- Variable files passed as arguments when executing the playbook
- Variables defined in Ansible roles

You can use Ansible vault to securely manage individual variables, entire files, or even structured data like YAML files. This data can then be safely stored in a version control system or shared with team members without exposing sensitive information.

IMPORTANT

Files are protected with symmetric encryption of the Advanced Encryption Standard (AES256), where a single password or passphrase is used both to encrypt and decrypt the data. Note that the way this is done has not been formally audited by a third party.

To simplify management, it makes sense to set up your Ansible project so that sensitive variables and all other variables are kept in separate files, or directories. Then you can protect the files containing sensitive variables with the **ansible-vault** command.

Creating an encrypted file

The following command prompts you for a new vault password. Then it opens a file for storing sensitive variables using the default editor.

```
# ansible-vault create vault.yml
New Vault password: <vault_password>
Confirm New Vault password: <vault_password>
```

Viewing an encrypted file

The following command prompts you for your existing vault password. Then it displays the sensitive contents of an already encrypted file.

```
# ansible-vault view vault.yml
Vault password: <vault_password>
my_secret: "yJJvPqhsiusmmPPZdnjndkdnYNDjdj782meUZcw"
```

Editing an encrypted file

The following command prompts you for your existing vault password. Then it opens the already encrypted file for you to update the sensitive variables using the default editor.

```
# ansible-vault edit vault.yml
Vault password: <vault_password>
```

Encrypting an existing file

The following command prompts you for a new vault password. Then it encrypts an existing unencrypted file.

```
# ansible-vault encrypt vault.yml
New Vault password: <vault_password>
Confirm New Vault password: <vault_password>
Encryption successful
```

Decrypting an existing file

The following command prompts you for your existing vault password. Then it decrypts an existing encrypted file.

```
# ansible-vault decrypt vault.yml
Vault password: <vault_password>
Decryption successful
```

Changing the password of an encrypted file

The following command prompts you for your original vault password, then for the new vault password.

```
# ansible-vault rekey vault.yml
Vault password: <vault_password>
New Vault password: <vault_password>
Confirm New Vault password: <vault_password>
Rekey successful
```

Basic application of Ansible vault variables in a playbook

```
---
- name: Create user accounts for all servers
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Create user from vault.yml file
      user:
        name: "{{ username }}"
        password: "{{ pwhash }}"
```

You read-in the file with variables (**vault.yml**) in the **vars_files** section of your Ansible Playbook, and you use the curly brackets the same way you would do with your ordinary variables. Then you either run the playbook with the **ansible-playbook --ask-vault-pass** command and you enter the password manually. Or you save the password in a separate file and you run the playbook with the **ansible-playbook --vault-password-file /path/to/my/vault-password-file** command.

Additional resources

- **ansible-vault(1)**, **ansible-playbook(1)** man pages on your system

- Ansible vault
- Ansible vault Best Practices

CHAPTER 4. JOINING RHEL SYSTEMS TO AN ACTIVE DIRECTORY BY USING RHEL SYSTEM ROLES

If your organization uses Microsoft Active Directory (AD) to centrally manage users, groups, and other resources, you can join your Red Hat Enterprise Linux (RHEL) host to this AD. For example, AD users can then log into RHEL and you can make services on the RHEL host available for authenticated AD users. By using the **ad_integration** RHEL system role, you can automate the integration of Red Hat Enterprise Linux system into an Active Directory (AD) domain.

> **NOTE**
>
> The **ad_integration** role is for deployments using direct AD integration without an Identity Management (IdM) environment. For IdM environments, use the **ansible-freeipa** roles.

4.1. JOINING RHEL TO AN ACTIVE DIRECTORY DOMAIN BY USING THE AD_INTEGRATION RHEL SYSTEM ROLE

You can use the **ad_integration** RHEL system role to automate the process of joining RHEL to an Active Directory (AD) domain.

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 managed node uses a DNS server that can resolve AD DNS entries.
- Credentials of an AD account which has permissions to join computers to the domain.
- Ensure that the required ports are open:
  - Ports required for direct integration of RHEL systems into AD using SSSD

Procedure

1. Store your sensitive variables in an encrypted file:

   a. Create the vault:

   ```
   $ ansible-vault create ~/vault.yml
   New Vault password: <vault_password>
   Confirm New Vault password: <vault_password>
   ```

   b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

   ```
   usr: administrator
   pwd: <password>
   ```

c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

```
---
- name: Active Directory integration
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Join an Active Directory
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.ad_integration
      vars:
        ad_integration_user: "{{ usr }}"
        ad_integration_password: "{{ pwd }}"
        ad_integration_realm: "ad.example.com"
        ad_integration_allow_rc4_crypto: false
        ad_integration_timesync_source: "time_server.ad.example.com"
```

The settings specified in the example playbook include the following:

ad_integration_allow_rc4_crypto: *<true|false>*

Configures whether the role activates the **AD-SUPPORT** crypto policy on the managed node. By default, RHEL does not support the weak RC4 encryption but, if Kerberos in your AD still requires RC4, you can enable this encryption type by setting **ad_integration_allow_rc4_crypto: true**.
Omit this the variable or set it to **false** if Kerberos uses AES encryption.

ad_integration_timesync_source: *<time_server>*

Specifies the NTP server to use for time synchronization. Kerberos requires a synchronized time among AD domain controllers and domain members to prevent replay attacks. If you omit this variable, the **ad_integration** role does not utilize the **timesync** RHEL system role to configure time synchronization on the managed node.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.ad_integration/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --ask-vault-pass --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Verification

- Check if AD users, such as **administrator**, are available locally on the managed node:

```
$ ansible managed-node-01.example.com -m command -a 'getent passwd
```

```
administrator@ad.example.com'
administrator@ad.example.com:*:1450400500:1450400513:Administrator:/home/administrator
@ad.example.com:/bin/bash
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.ad_integration/README.md** file
- **/usr/share/doc/rhel-system-roles/ad_integration/** directory
- Ansible vault

CHAPTER 5. CONFIGURING THE GRUB BOOT LOADER BY USING RHEL SYSTEM ROLES

By using the **bootloader** RHEL system role, you can automate the configuration and management tasks related to the GRUB boot loader.

This role currently supports configuring the GRUB boot loader, which runs on the following CPU architectures:

- AMD and Intel 64-bit architectures (x86-64)
- The 64-bit ARM architecture (ARMv8.0)
- IBM Power Systems, Little Endian (POWER9)

5.1. UPDATING THE EXISTING BOOT LOADER ENTRIES BY USING THE BOOTLOADER RHEL SYSTEM ROLE

You can use the **bootloader** RHEL system role to update the existing entries in the GRUB boot menu in an automated fashion. This way you can efficiently pass specific kernel command-line parameters that can optimize the performance or behavior of your systems.

For example, if you leverage systems, where detailed boot messages from the kernel and init system are not necessary, use **bootloader** to apply the **quiet** parameter to your existing boot loader entries on your managed nodes to achieve a cleaner, less cluttered, and more user-friendly booting experience.

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.
- You identified the kernel that corresponds to the boot loader entry you want to update.

Procedure

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

```
---
- name: Configuration and management of GRUB boot loader
  hosts: managed-node-01.example.com
  tasks:
    - name: Update existing boot loader entries
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.bootloader
      vars:
        bootloader_settings:
          - kernel:
              path: /boot/vmlinuz-5.14.0-362.24.1.el9_3.aarch64
            options:
```

```
          - name: quiet
            state: present
      bootloader_reboot_ok: true
```

The settings specified in the example playbook include the following:

kernel
: Specifies the kernel connected with the boot loader entry that you want to update.

options
: Specifies the kernel command-line parameters to update for your chosen boot loader entry (kernel).

bootloader_reboot_ok: *true*
: The role detects that a reboot is required for the changes to take effect and performs a restart of the managed node.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.bootloader/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Check that your specified boot loader entry has updated kernel command-line parameters:

```
# ansible managed-node-01.example.com -m ansible.builtin.command -a 'grubby --info=ALL'
managed-node-01.example.com | CHANGED | rc=0 >>
...
index=1
kernel="/boot/vmlinuz-5.14.0-362.24.1.el9_3.aarch64"
args="ro crashkernel=1G-4G:256M,4G-64G:320M,64G-:576M rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap $tuned_params quiet"
root="/dev/mapper/rhel-root"
initrd="/boot/initramfs-5.14.0-362.24.1.el9_3.aarch64.img $tuned_initrd"
title="Red Hat Enterprise Linux (5.14.0-362.24.1.el9_3.aarch64) 9.4 (Plow)"
id="2c9ec787230141a9b087f774955795ab-5.14.0-362.24.1.el9_3.aarch64"
...
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.bootloader/README.md** file
- **/usr/share/doc/rhel-system-roles/bootloader/** directory

- Working With Playbooks
- Using Variables
- Roles
- Configuring kernel command-line parameters

5.2. SECURING THE BOOT MENU WITH PASSWORD BY USING THE BOOTLOADER RHEL SYSTEM ROLE

You can use the **bootloader** RHEL system role to set a password to the GRUB boot menu in an automated fashion. This way you can efficiently prevent unauthorized users from modifying boot parameters, and to have better control over the system boot.

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.

Procedure

1. Store your sensitive variables in an encrypted file:
   a. Create the vault:

   ```
   $ ansible-vault create ~/vault.yml
   New Vault password: <vault_password>
   Confirm New Vault password: <vault_password>
   ```

   b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

   ```
   pwd: <password>
   ```

   c. Save the changes, and close the editor. Ansible encrypts the data in the vault.
2. Create a playbook file, for example **~/playbook.yml**, with the following content:

```
---
- name: Configuration and management of GRUB boot loader
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Set the bootloader password
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.bootloader
      vars:
        bootloader_password: "{{ pwd }}"
        bootloader_reboot_ok: true
```

The settings specified in the example playbook include the following:

bootloader_password: ***"{{ pwd }}"***

The variable ensures protection of boot parameters with a password.

bootloader_reboot_ok: ***true***

The role detects that a reboot is required for the changes to take effect and performs a restart of the managed node.

> **IMPORTANT**
>
> Changing the boot loader password is not an idempotent transaction. This means that if you apply the same Ansible playbook again, the result will not be the same, and the state of the managed node will change.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.bootloader/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check --ask-vault-pass ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Verification

1. On your managed node during the GRUB boot menu screen, press the **e** key for edit.

2. You are prompted for a username and a password:

```
Enter username:
root
Enter password:
_
```

Enter username: root

The boot loader username is always **root** and you do not need to specify it in your Ansible playbook.

Enter password: *<password>*

The boot loader password corresponds to the **pwd** variable that you defined in the **vault.yml** file.

3. You can view or edit configuration of the particular boot loader entry:

```
                         GRUB version 2.06

load_video
set gfxpayload=keep
insmod gzio
linux ($root)/vmlinuz-5.14.0-362.24.1.el9_3.aarch64 root=/dev/mapper/rhel-r\
oot ro crashkernel=1G-4G:256M,4G-64G:320M,64G-:576M rd.lvm.lv=rhel/root rd.\
lvm.lv=rhel/swap  quiet
initrd ($root)/initramfs-5.14.0-362.24.1.el9_3.aarch64.img $tuned_initrd

    Minimum Emacs-like screen editing is supported. TAB lists
    completions. Press Ctrl-x or F10 to boot, Ctrl-c or F2 for a
    command-line or ESC to discard edits and return to the GRUB menu.
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.bootloader/README.md** file
- **/usr/share/doc/rhel-system-roles/bootloader/** directory

5.3. SETTING A TIMEOUT FOR THE BOOT LOADER MENU BY USING THE BOOTLOADER RHEL SYSTEM ROLE

You can use the **bootloader** RHEL system role to configure a timeout for the GRUB boot loader menu in an automated way. You can update a period of time to intervene and select a non-default boot entry for various purposes.

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.

Procedure

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

```
---
- name: Configuration and management of the GRUB boot loader
  hosts: managed-node-01.example.com
  tasks:
    - name: Update the boot loader timeout
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.bootloader
      vars:
        bootloader_timeout: 10
```

   The settings specified in the example playbook include the following:

   bootloader_timeout: 10
   Input an integer to control for how long the GRUB boot loader menu is displayed before booting the default entry.

   For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.bootloader/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

1. Remotely restart your managed node:

```
# ansible managed-node-01.example.com -m ansible.builtin.reboot
managed-node-01.example.com | CHANGED => {
    "changed": true,
    "elapsed": 21,
    "rebooted": true
}
```

2. On the managed node, observe the GRUB boot menu screen.

The highlighted entry will be executed automatically in 10s

For how long this boot menu is displayed before GRUB automatically uses the default entry.

- Alternative: you can remotely query for the "timeout" settings in the **/boot/grub2/grub.cfg** file of your managed node:

```
# ansible managed-node-01.example.com -m ansible.builtin.command -a "grep
'timeout' /boot/grub2/grub.cfg"
managed-node-01.example.com | CHANGED | rc=0 >>
if [ x$feature_timeout_style = xy ] ; then
  set timeout_style=menu
  set timeout=10
# Fallback normal timeout code in case the timeout_style feature is
  set timeout=10
if [ x$feature_timeout_style = xy ] ; then
    set timeout_style=menu
    set timeout=10
    set orig_timeout_style=${timeout_style}
    set orig_timeout=${timeout}
      # timeout_style=menu + timeout=0 avoids the countdown code keypress check
      set timeout_style=menu
      set timeout=10
      set timeout_style=hidden
      set timeout=10
if [ x$feature_timeout_style = xy ]; then
  if [ "${menu_show_once_timeout}" ]; then
    set timeout_style=menu
    set timeout=10
    unset menu_show_once_timeout
    save_env menu_show_once_timeout
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.bootloader/README.md** file
- **/usr/share/doc/rhel-system-roles/bootloader/** directory

5.4. COLLECTING THE BOOT LOADER CONFIGURATION INFORMATION BY USING THE BOOTLOADER RHEL SYSTEM ROLE

You can use the **bootloader** RHEL system role to gather information about the GRUB boot loader entries in an automated fashion. You can use this information to verify the correct configuration of system boot parameters, such as kernel and initial RAM disk image paths.

As a result, you can for example:

- Prevent boot failures.
- Revert to a known good state when troubleshooting.
- Be sure that security-related kernel command-line parameters are correctly configured.

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.

Procedure

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

```
---
- name: Configuration and management of GRUB boot loader
  hosts: managed-node-01.example.com
  tasks:
    - name: Gather information about the boot loader configuration
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.bootloader
      vars:
        bootloader_gather_facts: true

    - name: Display the collected boot loader configuration information
      debug:
        var: bootloader_facts
```

   For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.bootloader/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- After you run the preceding playbook on the control node, you will see a similar command-line output as in the following example:

```
...
    "bootloader_facts": [
        {
            "args": "ro crashkernel=1G-4G:256M,4G-64G:320M,64G-:576M rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap $tuned_params quiet",
            "default": true,
            "id": "2c9ec787230141a9b087f774955795ab-5.14.0-362.24.1.el9_3.aarch64",
            "index": "1",
            "initrd": "/boot/initramfs-5.14.0-362.24.1.el9_3.aarch64.img $tuned_initrd",
            "kernel": "/boot/vmlinuz-5.14.0-362.24.1.el9_3.aarch64",
            "root": "/dev/mapper/rhel-root",
            "title": "Red Hat Enterprise Linux (5.14.0-362.24.1.el9_3.aarch64) 9.4 (Plow)"
        }
    ]
...
```

The command-line output shows the following notable configuration information about the boot entry:

args
: Command-line parameters passed to the kernel by the GRUB2 boot loader during the boot process. They configure various settings and behaviors of the kernel, initramfs, and other boot-time components.

id
: Unique identifier assigned to each boot entry in a boot loader menu. It consists of machine ID and the kernel version.

root
: The root filesystem for the kernel to mount and use as the primary filesystem during the boot.

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.bootloader/README.md** file
- **/usr/share/doc/rhel-system-roles/bootloader/** directory
- Understanding boot entries

CHAPTER 6. REQUESTING CERTIFICATES FROM A CA AND CREATING SELF-SIGNED CERTIFICATES BY USING RHEL SYSTEM ROLES

Many services, such as web servers, use TLS to encrypt connections with clients. These services require a private key and a certificate, and a trusted certificate authority (CA) which signs the certificate.

By using the **certificate** RHEL system role, you can automate the generation of private keys on managed nodes. Additionally, the role configures the **certmonger** service to send the certificate signing request (CSR) to a CA, and the service automatically renews the certificate before it expires.

For testing purposes, you can use the **certificate** role to create self-signed certificates instead of requesting a signed certificate from a CA.

6.1. REQUESTING A NEW CERTIFICATE FROM AN IDM CA BY USING THE CERTIFICATE RHEL SYSTEM ROLE

If a Red Hat Enterprise Linux host is a member of a RHEL Identity Management (IdM) environment, you can request TLS certificates from the IdM certificate authority (CA) and use them in the services that run on this host. By using the **certificate** RHEL system role, you can automate the process of creating a private key and letting the **certmonger** service request a certificate from the CA. By default, **certmonger** will also renew the certificate before it expires.

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 managed node is a member of an IdM domain and the domain uses the IdM-integrated CA.

Procedure

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

```
---
- name: Create certificates
  hosts: managed-node-01.example.com
  tasks:
    - name: Create a self-signed certificate
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.certificate
      vars:
        certificate_requests:
          - name: web-server
            ca: ipa
            dns: www.example.com
            principal: HTTP/www.example.com@EXAMPLE.COM
            run_before: systemctl stop httpd.service
            run_after: systemctl start httpd.service
```

The settings specified in the example playbook include the following:

name: *<path_or_file_name>*

Defines the name or path of the generated private key and certificate file:

- If you set the variable to **web-server**, the role stores the private key in the **/etc/pki/tls/private/web-server.key** and the certificate in the **/etc/pki/tls/certs/web-server.crt** files.
- If you set the variable to a path, such as **/tmp/web-server**, the role stores the private key in the **/tmp/web-server.key** and the certificate in the **/tmp/web-server.crt** files. Note that the directory you use must have the **cert_t** SELinux context set. You can use the **selinux** RHEL system role to manage SELinux contexts.

ca: ipa

Defines that the role requests the certificate from an IdM CA.

dns: *<hostname_or_list_of_hostnames>*

Sets the hostnames that the Subject Alternative Names (SAN) field in the issued certificate contains. You can use a wildcard (*) or specify multiple names in YAML list format.

principal: *<kerberos_principal>*

Optional: Sets the Kerberos principal that should be included in the certificate.

run_before: *<command>*

Optional: Defines a command that **certmonger** should execute before requesting the certificate from the CA.

run_after: *<command>*

Optional: Defines a command that **certmonger** should execute after it received the issued certificate from the CA.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.certificate/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- List the certificates that the **certmonger** service manages:

```
# ansible managed-node-01.example.com -m command -a 'getcert list'
...
Number of certificates and requests being tracked: 1.
Request ID '20240918142211':
        status: MONITORING
        stuck: no
```

```
    key pair storage: type=FILE,location='/etc/pki/tls/private/web-server.key'
    certificate: type=FILE,location='/etc/pki/tls/certs/web-server.crt'
    CA: IPA
    issuer: CN=Certificate Authority,O=EXAMPLE.COM
    subject: CN=www.example.com
    issued: 2024-09-18 16:22:11 CEST
    expires: 2025-09-18 16:22:10 CEST
    dns: www.example.com
    key usage: digitalSignature,keyEncipherment
    eku: id-kp-serverAuth,id-kp-clientAuth
    pre-save command: systemctl stop httpd.service
    post-save command: systemctl start httpd.service
    track: yes
    auto-renew: yes
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.certificate/README.md** file
- **/usr/share/doc/rhel-system-roles/certificate/** directory

6.2. REQUESTING A NEW SELF-SIGNED CERTIFICATE BY USING THE CERTIFICATE RHEL SYSTEM ROLE

If you require a TLS certificate for a test environment, you can use a self-signed certificate. By using the **certificate** RHEL system role, you can automate the process of creating a private key and letting the **certmonger** service create a self-signed certificate. By default, **certmonger** will also renew the certificate before it expires.

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.

Procedure

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

```
---
- name: Create certificates
  hosts: managed-node-01.example.com
  tasks:
    - name: Create a self-signed certificate
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.certificate
      vars:
        certificate_requests:
          - name: web-server
            ca: self-sign
            dns: test.example.com
```

The settings specified in the example playbook include the following:

name: *<path_or_file_name>*

Defines the name or path of the generated private key and certificate file:

- If you set the variable to **web-server**, the role stores the private key in the **/etc/pki/tls/private/web-server.key** and the certificate in the **/etc/pki/tls/certs/web-server.crt** files.
- If you set the variable to a path, such as **/tmp/web-server**, the role stores the private key in the **/tmp/web-server.key** and the certificate in the **/tmp/web-server.crt** files.
  Note that the directory you use must have the **cert_t** SELinux context set. You can use the **selinux** RHEL system role to manage SELinux contexts.

ca: self-sign

Defines that the role created a self-signed certificate.

dns: *<hostname_or_list_of_hostnames>*

Sets the hostnames that the Subject Alternative Names (SAN) field in the issued certificate contains. You can use a wildcard (*) or specify multiple names in YAML list format.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.certificate/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- List the certificates that the **certmonger** service manages:

```
# ansible managed-node-01.example.com -m command -a 'getcert list'
...
Number of certificates and requests being tracked: 1.
Request ID '20240918133610':
 status: MONITORING
 stuck: no
 key pair storage: type=FILE,location='/etc/pki/tls/private/web-server.key'
 certificate: type=FILE,location='/etc/pki/tls/certs/web-server.crt'
 CA: local
 issuer: CN=c32b16d7-5b1a4c5a-a953a711-c3ca58fb,CN=Local Signing Authority
 subject: CN=test.example.com
 issued: 2024-09-18 15:36:10 CEST
 expires: 2025-09-18 15:36:09 CEST
 dns: test.example.com
 key usage: digitalSignature,keyEncipherment
 eku: id-kp-serverAuth,id-kp-clientAuth
```

```
pre-save command:
post-save command:
track: yes
auto-renew: yes
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.certificate/README.md** file
- **/usr/share/doc/rhel-system-roles/certificate/** directory

CHAPTER 7. INSTALLING AND CONFIGURING WEB CONSOLE BY USING RHEL SYSTEM ROLES

With the **cockpit** RHEL system role, you can automatically deploy and enable the web console on multiple RHEL systems.

7.1. INSTALLING THE WEB CONSOLE BY USING THECOCKPIT RHEL SYSTEM ROLE

You can use the **cockpit** system role to automate installing and enabling the RHEL web console on multiple systems.

In this example, you use the **cockpit** system role to:

- Install the RHEL web console.
- Allow the **firewalld** and **selinux** system roles to configure the system for opening new ports.
- Set the web console to use a certificate from the **ipa** trusted certificate authority instead of using a self-signed certificate.

> **NOTE**
>
> You do not have to call the **firewall** or **certificate** system roles in the playbook to manage the firewall or create the certificate. The **cockpit** system role calls them automatically as needed.

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.

Procedure

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

```
---
- name: Manage the RHEL web console
  hosts: managed-node-01.example.com
  tasks:
    - name: Install RHEL web console
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.cockpit
      vars:
        cockpit_packages: default
        cockpit_manage_selinux: true
        cockpit_manage_firewall: true
        cockpit_certificates:
          - name: /etc/cockpit/ws-certs.d/01-certificate
            dns: ['localhost', 'www.example.com']
            ca: ipa
```

The settings specified in the example playbook include the following:

cockpit_manage_selinux: true
: Allow using the **selinux** system role to configure SELinux for setting up the correct port permissions on the **websm_port_t** SELinux type.

cockpit_manage_firewall: true
: Allow the **cockpit** system role to use the **firewalld** system role for adding ports.

cockpit_certificates: <YAML_dictionary>
: By default, the RHEL web console uses a self-signed certificate. Alternatively, you can add the **cockpit_certificates** variable to the playbook and configure the role to request certificates from an IdM certificate authority (CA) or to use an existing certificate and private key that is available on the managed node.
For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.cockpit/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.cockpit/README.md** file
- **/usr/share/doc/rhel-system-roles/cockpit** directory
- Requesting certificates using RHEL system roles

CHAPTER 8. SETTING A CUSTOM CRYPTOGRAPHIC POLICY BY USING RHEL SYSTEM ROLES

Custom cryptographic policies are a set of rules and configurations that manage the use of cryptographic algorithms and protocols. These policies help you to maintain a protected, consistent, and manageable security environment across multiple systems and applications.

By using the **crypto_policies** RHEL system role, you can quickly and consistently configure custom cryptographic policies across many operating systems in an automated fashion.

8.1. ENHANCING SECURITY WITH THE FUTURE CRYPTOGRAPHIC POLICY USING THE CRYPTO_POLICIES RHEL SYSTEM ROLE

You can use the **crypto_policies** RHEL system role to configure the **FUTURE** policy on your managed nodes. This policy helps to achieve for example:

- Future-proofing against emerging threats: anticipates advancements in computational power.
- Enhanced security: stronger encryption standards require longer key lengths and more secure algorithms.
- Compliance with high-security standards: for example in healthcare, telco, and finance the data sensitivity is high, and availability of strong cryptography is critical.

Typically, **FUTURE** is suitable for environments handling highly sensitive data, preparing for future regulations, or adopting long-term security strategies.

> **WARNING**
>
> Legacy systems or software does not have to support the more modern and stricter algorithms and protocols enforced by the **FUTURE** policy. For example, older systems might not support TLS 1.3 or larger key sizes. This could lead to compatibility problems.
>
> Also, using strong algorithms usually increases the computational workload, which could negatively affect your system performance.

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.

Procedure

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

```
---
- name: Configure cryptographic policies
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure the FUTURE cryptographic security policy on the managed node
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.crypto_policies
      vars:
        - crypto_policies_policy: FUTURE
        - crypto_policies_reboot_ok: true
```

The settings specified in the example playbook include the following:

crypto_policies_policy: *FUTURE*

Configures the required cryptographic policy (**FUTURE**) on the managed node. It can be either the base policy or a base policy with some sub-policies. The specified base policy and sub-policies have to be available on the managed node. The default value is **null**. It means that the configuration is not changed and the **crypto_policies** RHEL system role will only collect the Ansible facts.

crypto_policies_reboot_ok: *true*

Causes the system to reboot after the cryptographic policy change to make sure all of the services and applications will read the new configuration files. The default value is **false**.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.crypto_policies/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

> **WARNING**
>
> Because the **FIPS:OSPP** system-wide subpolicy contains further restrictions for cryptographic algorithms required by the Common Criteria (CC) certification, the system is less interoperable after you set it. For example, you cannot use RSA and DH keys shorter than 3072 bits, additional SSH algorithms, and several TLS groups. Setting **FIPS:OSPP** also prevents connecting to Red Hat Content Delivery Network (CDN) structure. Furthermore, you cannot integrate Active Directory (AD) into the IdM deployments that use **FIPS:OSPP**, communication between RHEL hosts using **FIPS:OSPP** and AD domains might not work, or some AD accounts might not be able to authenticate.
>
> Note that your **system is not CC-compliant** after you set the **FIPS:OSPP** cryptographic subpolicy. The only correct way to make your RHEL system compliant with the CC standard is by following the guidance provided in the **cc-config** package. See the Common Criteria section on the Product compliance Red Hat Customer Portal page for a list of certified RHEL versions, validation reports, and links to CC guides hosted at the National Information Assurance Partnership (NIAP) website.

Verification

1. On the control node, create another playbook named, for example, **verify_playbook.yml**:

```
---
- name: Verification
  hosts: managed-node-01.example.com
  tasks:
    - name: Verify active cryptographic policy
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.crypto_policies
    - name: Display the currently active cryptographic policy
      ansible.builtin.debug:
        var: crypto_policies_active
```

   The settings specified in the example playbook include the following:

   crypto_policies_active
   : An exported Ansible fact that contains the currently active policy name in the format as accepted by the **crypto_policies_policy** variable.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/verify_playbook.yml
```

3. Run the playbook:

```
$ ansible-playbook ~/verify_playbook.yml
TASK [debug] **************************
ok: [host] => {
```

```
    "crypto_policies_active": "FUTURE"
}
```

The **crypto_policies_active** variable shows the active policy on the managed node.

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.crypto_policies/README.md** file
- **/usr/share/doc/rhel-system-roles/crypto_policies/** directory
- **update-crypto-policies(8)** and **crypto-policies(7)** manual pages

CHAPTER 9. RESTRICTING THE EXECUTION OF APPLICATIONS BY USING THE FAPOLICYD RHEL SYSTEM ROLE

By using the **fapolicyd** software framework, you can restrict the execution of applications based on a user-defined policy and the framework verifies the integrity of applications before execution. This an efficient method to prevent running untrustworthy and possibly malicious applications. You can automate the installation and configuration of **fapolicyd** by using the **fapolicyd** RHEL system role.

> **IMPORTANT**
>
> The **fapolicyd** service prevents only the execution of unauthorized applications that run as regular users, and not as **root**.

9.1. PREVENTING USERS FROM EXECUTING UNTRUSTWORTHY CODE BY USING THE FAPOLICYD RHEL SYSTEM ROLE

You can automate the installation and configuration of the **fapolicyd** service by using the **fapolicyd** RHEL system role. With this role, you can remotely configure the service to allow users to execute only trusted applications, for example, the ones which are listed in the RPM database and in an allow list. Additionally, the service can perform integrity checks before it executes an allowed application.

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.

Procedure

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

```
---
- name: Configuring fapolicyd
  hosts: managed-node-01.example.com
  tasks:
    - name: Allow only executables installed from RPM database and specific files
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.fapolicyd
      vars:
        fapolicyd_setup_permissive: false
        fapolicyd_setup_integrity: sha256
        fapolicyd_setup_trust: rpmdb,file
        fapolicyd_add_trusted_file:
          - <path_to_allowed_command>
          - <path_to_allowed_service>
```

The settings specified in the example playbook include the following:

fapolicyd_setup_permissive: *<true|false>*

Enables or disables sending policy decisions to the kernel for enforcement. Set this variable for debugging and testing purposes to **false**.

fapolicyd_setup_integrity: *<type_type>*

Defines the integrity checking method. You can set one of the following values:

- **none** (default): Disables integrity checking.
- **size**: The service compares only the file sizes of allowed applications.
- **ima**: The service checks the SHA-256 hash that the kernel's Integrity Measurement Architecture (IMA) stored in a file's extended attribute. Additionally, the service performs a size check. Note that the role does not configure the IMA kernel subsystem. To use this option, you must manually configure the IMA subsystem.
- **sha256**: The service compares the SHA-256 hash of allowed applications.

fapolicyd_setup_trust: *<trust_backends>*

Defines the list of trust backends. If you include the **file** backend, specify the allowed executable files in the **fapolicyd_add_trusted_file** list.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.fapolicyd.README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook ~/playbook.yml --syntax-check
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Execute a binary application that is not on the allow list as a user:

```
$ ansible managed-node-01.example.com -m command -a 'su -c
"/bin/not_authorized_application " <user_name>'
bash: line 1: /bin/not_authorized_application: Operation not permitted non-zero return code
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.fapolicyd/README.md** file
- **/usr/share/doc/rhel-system-roles/fapolicyd/** directory

CHAPTER 10. CONFIGURING FIREWALLD BY USING RHEL SYSTEM ROLES

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 at once.

The **rhel-system-roles** package contains the **rhel-system-roles.firewall** RHEL system role. This role was introduced for automated configurations of the **firewalld** service.

With the **firewall** RHEL system 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

10.1. RESETTING THE FIREWALLD SETTINGS BY USING THE FIREWALL RHEL SYSTEM ROLE

Over time, updates to your firewall configuration can accumulate to the point, where they could lead to unintended security risks. With the **firewall** RHEL system role, you can reset the **firewalld** settings to their default state in an automated fashion. This way you can efficiently remove any unintentional or insecure firewall rules and simplify their management.

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.

Procedure

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

```
---
- name: Reset firewalld example
  hosts: managed-node-01.example.com
  tasks:
    - name: Reset firewalld
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.firewall
      vars:
        firewall:
          - previous: replaced
```

The settings specified in the example playbook include the following:

previous: replaced

Removes all existing user-defined settings and resets the **firewalld** settings to defaults. If you combine the **previous:replaced** parameter with other settings, the **firewall** role removes all existing settings before applying new ones.
For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.firewall/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Run this command on the control node to remotely check that all firewall configuration on your managed node was reset to its default values:

```
# ansible managed-node-01.example.com -m ansible.builtin.command -a 'firewall-cmd --list-all-zones'
```

Additional resources

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

10.2. FORWARDING INCOMING TRAFFIC IN FIREWALLD FROM ONE LOCAL PORT TO A DIFFERENT LOCAL PORT BY USING THE FIREWALL RHEL SYSTEM ROLE

You can use the **firewall** RHEL system role to remotely configure forwarding of incoming traffic from one local port to a different local port.

For example, if you have an environment where multiple services co-exist on the same machine and need the same default port, there are likely to become port conflicts. These conflicts can disrupt services and cause a downtime. With the **firewall** RHEL system role, you can efficiently forward traffic to alternative ports to ensure that your services can run simultaneously without modification to their configuration.

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.

Procedure

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

```
---
- name: Configure firewalld
  hosts: managed-node-01.example.com
  tasks:
    - name: Forward incoming traffic on port 8080 to 443
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.firewall
      vars:
        firewall:
          - forward_port: 8080/tcp;443;
            state: enabled
            runtime: true
            permanent: true
```

The settings specified in the example playbook include the following:

forward_port: 8080/tcp;443

Traffic coming to the local port 8080 using the TCP protocol is forwarded to the port 443.

runtime: true

Enables changes in the runtime configuration. The default is set to **true**.
For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.firewall/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- On the control node, run the following command to remotely check the forwarded-ports on your managed node:

```
# ansible managed-node-01.example.com -m ansible.builtin.command -a 'firewall-cmd --list-forward-ports'
managed-node-01.example.com | CHANGED | rc=0 >>
port=8080:proto=tcp:toport=443:toaddr=
```

Additional resources

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

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

As a system administrator, you can use the **firewall** RHEL 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.

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.

Procedure

1. Create a playbook file, for example ~/**playbook.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
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.firewall
      vars:
        firewall:
          - zone: dmz
            interface: enp1s0
            service: https
            state: enabled
            runtime: true
            permanent: true
```

   For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.firewall/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- On the control node, run the following command to remotely check the information about the **dmz** zone on your managed node:

```
# ansible managed-node-01.example.com -m ansible.builtin.command -a 'firewall-cmd
--zone=dmz --list-all'
managed-node-01.example.com | CHANGED | rc=0 >>
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** file
- **/usr/share/doc/rhel-system-roles/firewall/** directory

CHAPTER 11. CONFIGURING A HIGH-AVAILABILITY CLUSTER BY USING RHEL 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.

11.1. VARIABLES OF THE HA_CLUSTER RHEL SYSTEM ROLE

In an **ha_cluster** RHEL 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** RHEL 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** RHEL system role. When this variable is set to **true**, the default value, 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_enable_repos_resilient_storage

(RHEL 8.10 and later) A boolean flag that enables the repositories containing resilient storage packages, such as **dlm** or **gfs2**. For this option to take effect, **ha_cluster_enable_repos** must be set to **true**. The default value of this variable is **false**.

ha_cluster_manage_firewall

(RHEL 8.8 and later) A boolean flag that determines whether the **ha_cluster** RHEL system role manages the firewall. When **ha_cluster_manage_firewall** is set to **true**, the firewall high availability service and the **fence-virt** port are enabled. When **ha_cluster_manage_firewall** is set to **false**, the **ha_cluster** RHEL system role does not manage the firewall. If your system is running the **firewalld** service, you must set the parameter to **true** in your playbook.
You can use the **ha_cluster_manage_firewall** parameter to add ports, but you cannot use the parameter to remove ports. To remove ports, use the **firewall** system role directly.

In RHEL 8.8 and later, the firewall is no longer configured by default, because it is configured only when **ha_cluster_manage_firewall** is set to **true**.

ha_cluster_manage_selinux

(RHEL 8.8 and later) A boolean flag that determines whether the **ha_cluster** RHEL system role manages the ports belonging to the firewall high availability service using the **selinux** RHEL system role. When **ha_cluster_manage_selinux** is set to **true**, the ports belonging to the firewall high availability service are associated with the SELinux port type **cluster_port_t**. When **ha_cluster_manage_selinux** is set to **false**, the **ha_cluster** RHEL system role does not manage SELinux.
If your system is running the **selinux** service, you must set this parameter to **true** in your playbook. Firewall configuration is a prerequisite for managing SELinux. If the firewall is not installed, the managing SELinux policy is skipped.

You can use the **ha_cluster_manage_selinux** parameter to add policy, but you cannot use the parameter to remove policy. To remove policy, use the **selinux** RHEL system role directly.

ha_cluster_cluster_present

A boolean flag which, if set to **true**, 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 playbook and not supported by the role will be lost.

If **ha_cluster_cluster_present** is set to **false**, all HA cluster configuration will be removed from the target hosts.

The default value of this variable is **true**.

The following example playbook removes all cluster configuration on **node1** and **node2**

```
- hosts: node1 node2
  vars:
    ha_cluster_cluster_present: false

  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 **true**.

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. To protect sensitive data, 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_hacluster_qdevice_password

(RHEL 8.9 and later) A string value that specifies the password of the **hacluster** user for a quorum device. This parameter is needed only if the **ha_cluster_quorum** parameter is configured to use a quorum device of type **net** and the password of the **hacluster** user on the quorum device is different from the password of the **hacluster** user specified with the **ha_cluster_hacluster_password** parameter. The **hacluster** user has full access to a cluster. To protect sensitive data, vault encrypt the password, as described in Encrypting content with Ansible Vault . There is no default value for this password.

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** RHEL 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_pcsd_certificates

(RHEL 8.8 and later) Creates a **pcsd** private key and certificate using the **certificate** RHEL system role.
If your system is not configured with a **pcsd** private key and certificate, you can create them in one of two ways:

- Set the **ha_cluster_pcsd_certificates** variable. When you set the **ha_cluster_pcsd_certificates** variable, the **certificate** RHEL system role is used internally and it creates the private key and certificate for **pcsd** as defined.

- Do not set the **ha_cluster_pcsd_public_key_src**, **ha_cluster_pcsd_private_key_src**, or the **ha_cluster_pcsd_certificates** variables. If you do not set any of these variables, the **ha_cluster** RHEL system role will create **pcsd** certificates by means of **pcsd** itself. The value of **ha_cluster_pcsd_certificates** is set to the value of the variable **certificate_requests** as specified in the **certificate** RHEL system role. For more information about the **certificate** RHEL system role, see Requesting certificates using RHEL system roles.

The following operational considerations apply to the use of the **ha_cluster_pcsd_certificate** variable:

- Unless you are using IPA and joining the systems to an IPA domain, the **certificate** RHEL system role creates self-signed certificates. In this case, you must explicitly configure trust settings outside of the context of RHEL system roles. System roles do not support configuring trust settings.

- When you set the **ha_cluster_pcsd_certificates** variable, do not set the **ha_cluster_pcsd_public_key_src** and **ha_cluster_pcsd_private_key_src** variables.

- When you set the **ha_cluster_pcsd_certificates** variable, **ha_cluster_regenerate_keys** is ignored for this certificate - key pair.

The default value of this variable is **[]**.
For an example **ha_cluster** RHEL system role playbook that creates TLS certificates and key files in a high availability cluster, see Creating pcsd TLS certificates and key files for a high availability cluster.

ha_cluster_regenerate_keys

A boolean flag which, when set to **true**, determines that pre-shared keys and TLS certificates will be regenerated. For more information about 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 **false**.

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** RHEL 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** RHEL 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 following items for cluster quorum:

- **options** (optional) - List of name-value dictionaries configuring quorum. Allowed options are: **auto_tie_breaker**, **last_man_standing**, **last_man_standing_window**, and **wait_for_all**. For information about quorum options, see the **votequorum**(5) man page.

- **device** (optional) - (RHEL 8.8 and later) Configures the cluster to use a quorum device. By default, no quorum device is used.

  - **model** (mandatory) - Specifies a quorum device model. Only **net** is supported

  - **model_options** (optional) - List of name-value dictionaries configuring the specified quorum device model. For model **net**, you must specify **host** and **algorithm** options.
    Use the **pcs-address** option to set a custom **pcsd** address and port to connect to the **qnetd** host. If you do not specify this option, the role connects to the default **pcsd** port on the **host**.

- **generic_options** (optional) - List of name-value dictionaries setting quorum device options that are not model specific.
- **heuristics_options** (optional) - List of name-value dictionaries configuring quorum device heuristics.
  For information about quorum device options, see the **corosync-qdevice**(8) man page. The generic options are **sync_timeout** and **timeout**. For model **net** options see the **quorum.device.net** section. For heuristics options, see the **quorum.device.heuristics** section.

  To regenerate a quorum device TLS certificate, set the **ha_cluster_regenerate_keys** variable to **true**.

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
  device:
    model: string
    model_options:
      - name: option1_name
        value: option1_value
      - name: option2_name
        value: option2_value
    generic_options:
      - name: option1_name
        value: option1_value
      - name: option2_name
        value: option2_value
    heuristics_options:
      - name: option1_name
        value: option1_value
      - name: option2_name
        value: option2_value
```

For an example **ha_cluster** RHEL system role playbook that configures cluster quorum, see Configuring Corosync values in a high availability cluster . For an example **ha_cluster** RHEL system role playbook that configures a cluster using a quorum device, see Configuring a high availability cluster using a quorum device.

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 **false**.
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. For information about these options, see the **Configuration via environment** section of the **sbd**(8) man page.
Supported options are:

- **delay-start** - defaults to **false**, documented as **SBD_DELAY_START**
- **startmode** - defaults to **always**, documented as **SBD_START_MODE**
- **timeout-action** - defaults to **flush,reboot**, documented as **SBD_TIMEOUT_ACTION**
- **watchdog-timeout** - defaults to **5**, documented as **SBD_WATCHDOG_TIMEOUT**

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 about 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_node_options

(RHEL 8. 10 and later) This variable defines settings which vary from one cluster node to another. It sets the options for the specified nodes, but does not specify which nodes form the cluster. You specify which nodes form the cluster with the **hosts** parameter in an inventory or a playbook.
The items you configure with this variable are as follows:

- **node_name** (mandatory) - Name of the node for which to define Pacemaker node attributes. It must match a name defined for a node.

- **attributes** (optional) - List of sets of Pacemaker node attributes for the node. Currently, only one set is supported. The first set is used and the rest are ignored.

The structure of the **ha_cluster_node_options** variable is as follows:

```
ha_cluster_node_options:
  - node_name: node1
    attributes:
      - attrs:
          - name: attribute1
            value: value1_node1
          - name: attribute2
            value: value2_node1
  - node_name: node2
    attributes:
      - attrs:
          - name: attribute1
            value: value1_node2
          - name: attribute2
            value: value2_node2
```

By default, no node options are defined.

For an example **ha_cluster** RHEL system role playbook that includes node options configuration, see Configuring a high availability cluster with node attributes .

ha_cluster_resource_primitives

This variable defines pacemaker resources configured by the RHEL system role, including fencing resources. You can configure the following items for each resource:

- **id** (mandatory) - ID of a resource.

- **agent** (mandatory) - Name of a resource or fencing 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 fencing agent.

- **meta_attrs** (optional) - List of sets of the resource’s meta attributes. Currently, only one set is supported.

- **copy_operations_from_agent** (optional) - (RHEL 8.9 and later) Resource agents usually define default settings for resource operations, such as **interval** and **timeout**, optimized for the specific agent. If this variable is set to **true**, then those settings are copied to the resource configuration. Otherwise, clusterwide defaults apply to the resource. If you also define resource operation defaults for the resource with the **ha_cluster_resource_operation_defaults** role variable, you can set this to **false**. The default value of this variable is **true**.

- **operations** (optional) - List of the resource’s operations.

- **action** (mandatory) - Operation action as defined by pacemaker and the resource or fencing 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** RHEL 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
  copy_operations_from_agent: bool
  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** RHEL 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. You can configure the following items for each resource group:

- **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** RHEL 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** RHEL 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. You can configure the following items for a resource clone:

- **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 **true** or **false**.
- **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** RHEL system role is as follows:

```
ha_cluster_resource_clones:
  - resource_id: resource-to-be-cloned
    promotable: true
    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** RHEL system role playbook that includes resource clone configuration, see Configuring a high availability cluster with fencing and resources .

ha_cluster_resource_defaults

(RHEL 8.9 and later) This variable defines sets of resource defaults. You can define multiple sets of defaults and apply them to resources of specific agents using rules. The defaults you specify with the **ha_cluster_resource_defaults** variable do not apply to resources which override them with their own defined values.
Only meta attributes can be specified as defaults.

You can configure the following items for each defaults set:

- **id** (optional) - ID of the defaults set. If not specified, it is autogenerated.
- **rule** (optional) - Rule written using **pcs** syntax defining when and for which resources the set applies. For information on specifying a rule, see the **resource defaults set create** section of the **pcs**(8) man page.
- **score** (optional) - Weight of the defaults set.
- **attrs** (optional) - Meta attributes applied to resources as defaults.

The structure of the **ha_cluster_resource_defaults** variable is as follows:

```
ha_cluster_resource_defaults:
  meta_attrs:
    - id: defaults-set-1-id
      rule: rule-string
      score: score-value
      attrs:
        - name: meta_attribute1_name
          value: meta_attribute1_value
        - name: meta_attribute2_name
          value: meta_attribute2_value
    - id: defaults-set-2-id
      rule: rule-string
      score: score-value
      attrs:
        - name: meta_attribute3_name
          value: meta_attribute3_value
        - name: meta_attribute4_name
          value: meta_attribute4_value
```

For an example **ha_cluster** RHEL system role playbook that configures resource defaults, see Configuring a high availability cluster with resource and resource operation defaults .

ha_cluster_resource_operation_defaults

(RHEL 8.9 and later) This variable defines sets of resource operation defaults. You can define multiple sets of defaults and apply them to resources of specific agents and specific resource operations using rules. The defaults you specify with the **ha_cluster_resource_operation_defaults** variable do not apply to resource operations which override them with their own defined values. By default, the **ha_cluster** RHEL system role configures resources to define their own values for resource operations. For information about overriding these defaults with the **ha_cluster_resource_operations_defaults** variable, see the description of the **copy_operations_from_agent** item in **ha_cluster_resource_primitives**.
Only meta attributes can be specified as defaults.

The structure of the **ha_cluster_resource_operations_defaults** variable is the same as the structure for the **ha_cluster_resource_defaults** variable, with the exception of how you specify a rule. For information about specifying a rule to describe the resource operation to which a set applies, see the **resource op defaults set create** section of the **pcs**(8) man page.

ha_cluster_stonith_levels

(RHEL 8.10 and later) This variable defines STONITH levels, also known as fencing topology. Fencing levels configure a cluster to use multiple devices to fence nodes. You can define alternative devices in case one device fails and you can require multiple devices to all be executed successfully to consider a node successfully fenced. For more information on fencing levels, see Configuring fencing levels in Configuring and managing high availability clusters.
You can configure the following items when defining fencing levels:

- **level** (mandatory) - Order in which to attempt the fencing level. Pacemaker attempts levels in ascending order until one succeeds.
- **target** (optional) - Name of a node this level applies to.
- You must specify one of the following three selections:
  - **target_pattern** - POSIX extended regular expression matching the names of the nodes this level applies to.
  - **target_attribute** - Name of a node attribute that is set for the node this level applies to.
  - **target_attribute** and **target_value** - Name and value of a node attribute that is set for the node this level applies to.
- **resouce_ids** (mandatory) - List of fencing resources that must all be tried for this level.
  By default, no fencing levels are defined.

The structure of the fencing levels definition that you configure with the **ha_cluster** RHEL system role is as follows:

```
ha_cluster_stonith_levels:
  - level: 1..9
    target: node_name
    target_pattern: node_name_regular_expression
    target_attribute: node_attribute_name
    target_value: node_attribute_value
    resource_ids:
      - fence_device_1
      - fence_device_2
  - level: 1..9
    target: node_name
    target_pattern: node_name_regular_expression
    target_attribute: node_attribute_name
    target_value: node_attribute_value
    resource_ids:
      - fence_device_1
      - fence_device_2
```

For an example **ha_cluster** RHEL system role playbook that configures fencing defaults, see Configuring a high availability cluster with fencing levels .

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.
You can configure the following items for a resource location constraint:

- **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
```

You can configure the following items for a resource location constraint that specifies a resource ID and a rule:

- **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** RHEL 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.
You can configure the following items for a simple resource colocation constraint:

- **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
```

You can configure the following items for a resource set colocation constraint:

- **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** RHEL 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.
You can configure the following items for a simple resource order constraint:

- **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
```

You can configure the following items for a resource set order constraint:

- **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** RHEL 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.
You can configure the following items for a simple resource ticket constraint:

- **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
```

You can configure the following items for a resource set ticket constraint:

- **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** RHEL system role playbook that creates a cluster with resource constraints, see Configuring a high availability cluster with resource constraints .

ha_cluster_qnetd

(RHEL 8.8 and later) This variable configures a **qnetd** host which can then serve as an external quorum device for clusters.
You can configure the following items for a **qnetd** host:

- **present** (optional) - If **true**, configure a **qnetd** instance on the host. If **false**, remove **qnetd** configuration from the host. The default value is **false**. If you set this **true**, you must set **ha_cluster_cluster_present** to **false**.
- **start_on_boot** (optional) - Configures whether the **qnetd** instance should start automatically on boot. The default value is **true**.
- **regenerate_keys** (optional) - Set this variable to **true** to regenerate the **qnetd** TLS certificate. If you regenerate the certificate, you must either re-run the role for each cluster to connect it to the **qnetd** host again or run **pcs** manually.

You cannot run **qnetd** on a cluster node because fencing would disrupt **qnetd** operation.
For an example **ha_cluster** RHEL system role playbook that configures a cluster using a quorum device, see Configuring a cluster using a quorum device .

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** file
- **/usr/share/doc/rhel-system-roles/ha_cluster/** directory

11.2. SPECIFYING AN INVENTORY FOR THE HA_CLUSTER RHEL SYSTEM ROLE

When configuring an HA cluster using the **ha_cluster** RHEL system role playbook, you configure the names and addresses of the nodes for the cluster in an inventory.

11.2.1. 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. The order of the addresses must be the same for all nodes, so that the addresses belonging to a particular link are specified in the same position for all nodes.

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
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** file
- **/usr/share/doc/rhel-system-roles/ha_cluster/** directory

11.2.2. 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 accessible from all nodes, each node can use different names for the devices. Watchdog devices can be different for each node as well. For information about 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 Variables of the **ha_cluster** RHEL system role.

For each node in an inventory, you can optionally specify the following items:

- **sbd_watchdog_modules** (optional) - (RHEL 8.9 and later) Watchdog kernel modules to be loaded, which create **/dev/watchdog*** devices. Defaults to empty list if not set.
- **sbd_watchdog_modules_blocklist** (optional) - (RHEL 8.9 and later) Watchdog kernel modules to be unloaded and blocked. Defaults to empty list if not set.
- **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. Always refer to the devices using the long, stable device name (/dev/disk/by-id/).

The following example shows an inventory that configures watchdog and SBD devices for targets **node1** and **node2**.

```
all:
  hosts:
    node1:
      ha_cluster:
        sbd_watchdog_modules:
          - module1
          - module2
        sbd_watchdog: /dev/watchdog2
        sbd_devices:
          - /dev/disk/by-id/000001
          - /dev/disk/by-id/000001
          - /dev/disk/by-id/000003
    node2:
      ha_cluster:
        sbd_watchdog_modules:
          - module1
        sbd_watchdog_modules_blocklist:
          - module2
        sbd_watchdog: /dev/watchdog1
        sbd_devices:
          - /dev/disk/by-id/000001
          - /dev/disk/by-id/000002
          - /dev/disk/by-id/000003
```

For an example procedure that creates high availability cluster that uses SBD fencing, see Configuring a high availability cluster with SBD node fencing.

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** file
- **/usr/share/doc/rhel-system-roles/ha_cluster/** directory

11.3. CREATING PCSD TLS CERTIFICATES AND KEY FILES FOR A HIGH AVAILABILITY CLUSTER

(RHEL 8.8 and later) The connection between cluster nodes is secured using Transport Layer Security (TLS) encryption. By default, the **pcsd** daemon generates self-signed certificates. For many deployments, however, you may want to replace the default certificates with certificates issued by a certificate authority of your company and apply your company certificate policies for **pcsd**.

You can use the **ha_cluster** RHEL system role to create TLS certificates and key files in a high availability cluster. When you run this playbook, the **ha_cluster** RHEL system role uses the **certificate** RHEL system role internally to manage TLS certificates.

> **WARNING**
>
> The **ha_cluster** RHEL system role replaces any existing cluster configuration on the specified nodes. Any settings not specified in the playbook will be lost.

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 systems that you will use as your cluster members have active subscription coverage for RHEL and the RHEL High Availability Add-On.
- The inventory file specifies the cluster nodes as described in Specifying an inventory for the ha_cluster RHEL system role. For general information about creating an inventory file, see Preparing a control node on RHEL 8 .

Procedure

1. Store your sensitive variables in an encrypted file:
   a. Create the vault:

```
$ ansible-vault create ~/vault.yml
New Vault password: <vault_password>
Confirm New Vault password: <vault_password>
```

   b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

```
cluster_password: <cluster_password>
```

   c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

```
---
- name: Create a high availability cluster
  hosts: node1 node2
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Create TLS certificates and key files in a high availability cluster
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.ha_cluster
      vars:
        ha_cluster_cluster_name: my-new-cluster
        ha_cluster_hacluster_password: "{{ cluster_password }}"
```

```
ha_cluster_manage_firewall: true
ha_cluster_manage_selinux: true
ha_cluster_pcsd_certificates:
  - name: FILENAME
    common_name: "{{ ansible_hostname }}"
    ca: self-sign
```

The settings specified in the example playbook include the following:

ha_cluster_cluster_name: *<cluster_name>*
: The name of the cluster you are creating.

ha_cluster_hacluster_password: *<password>*
: The password of the **hacluster** user. The **hacluster** user has full access to a cluster.

ha_cluster_manage_firewall: true
: A variable that determines whether the **ha_cluster** RHEL system role manages the firewall.

ha_cluster_manage_selinux: true
: A variable that determines whether the **ha_cluster** RHEL system role manages the ports of the firewall high availability service using the **selinux** RHEL system role.

ha_cluster_pcsd_certificates: *<certificate_properties>*
: A variable that creates a self-signed **pcsd** certificate and private key files in **/var/lib/pcsd**. In this example, the **pcsd** certificate has the file name **FILENAME.crt** and the key file is named **FILENAME.key**.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check --ask-vault-pass ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** file
- **/usr/share/doc/rhel-system-roles/ha_cluster/** directory
- Requesting certificates using RHEL system roles

11.4. CONFIGURING A HIGH AVAILABILITY CLUSTER RUNNING NO RESOURCES

You can use the **ha_cluster** system role to configure a basic cluster in a simple, automatic way. Once you have created a basic cluster, you can use the **pcs** command-line interface to configure the other cluster components and behaviors on a resource-by-resource basis. The following example procedure configures a basic two-node cluster with no fencing configured using the minimum required parameters.

> **WARNING**
>
> The **ha_cluster** system role replaces any existing cluster configuration on the specified nodes. Any settings not specified in the playbook will be lost.

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 systems that you will use as your cluster members have active subscription coverage for RHEL and the RHEL High Availability Add-On.
- The inventory file specifies the cluster nodes as described in Specifying an inventory for the ha_cluster system role. For general information about creating an inventory file, see Preparing a control node on RHEL 8.

Procedure

1. Store your sensitive variables in an encrypted file:
    a. Create the vault:
    ```
    $ ansible-vault create ~/vault.yml
    New Vault password: <vault_password>
    Confirm New Vault password: <vault_password>
    ```
    b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:
    ```
    cluster_password: <cluster_password>
    ```
    c. Save the changes, and close the editor. Ansible encrypts the data in the vault.
2. Create a playbook file, for example **~/playbook.yml**, with the following content:
```
---
- name: Create a high availability cluster
  hosts: node1 node2
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Create cluster with minimum required parameters and no fencing
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.ha_cluster
```

```
vars:
  ha_cluster_cluster_name: my-new-cluster
  ha_cluster_hacluster_password: "{{ cluster_password }}"
  ha_cluster_manage_firewall: true
  ha_cluster_manage_selinux: true
```

The settings specified in the example playbook include the following:

ha_cluster_cluster_name: *<cluster_name>*
: The name of the cluster you are creating.

ha_cluster_hacluster_password: *<password>*
: The password of the **hacluster** user. The **hacluster** user has full access to a cluster.

ha_cluster_manage_firewall: true
: A variable that determines whether the **ha_cluster** RHEL system role manages the firewall.

ha_cluster_manage_selinux: true
: A variable that determines whether the **ha_cluster** RHEL system role manages the ports of the firewall high availability service using the **selinux** RHEL system role.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check --ask-vault-pass ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** file
- **/usr/share/doc/rhel-system-roles/ha_cluster/** directory

11.5. CONFIGURING A HIGH AVAILABILITY CLUSTER WITH FENCING AND RESOURCES

The specific components of a cluster configuration depend on your individual needs, which vary between sites. The following example procedure shows the formats for configuring different cluster components by using the **ha_cluster** RHEL system role. The configured cluster includes a fencing device, cluster resources, resource groups, and a cloned resource.

> **WARNING**
>
> The **ha_cluster** RHEL system role replaces any existing cluster configuration on the specified nodes. Any settings not specified in the playbook will be lost.

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 systems that you will use as your cluster members have active subscription coverage for RHEL and the RHEL High Availability Add-On.
- The inventory file specifies the cluster nodes as described in Specifying an inventory for the ha_cluster RHEL system role. For general information about creating an inventory file, see Preparing a control node on RHEL 8 .

Procedure

1. Store your sensitive variables in an encrypted file:
   a. Create the vault:

      ```
      $ ansible-vault create ~/vault.yml
      New Vault password: <vault_password>
      Confirm New Vault password: <vault_password>
      ```

   b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

      ```
      cluster_password: <cluster_password>
      ```

   c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

```
---
- name: Create a high availability cluster
  hosts: node1 node2
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Create cluster with fencing and resources
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.ha_cluster
      vars:
        ha_cluster_cluster_name: my-new-cluster
        ha_cluster_hacluster_password: "{{ cluster_password }}"
```

```
ha_cluster_manage_firewall: true
ha_cluster_manage_selinux: true
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
```

The settings specified in the example playbook include the following:

ha_cluster_cluster_name: *<cluster_name>*
: The name of the cluster you are creating.

ha_cluster_hacluster_password: *<password>*
: The password of the **hacluster** user. The **hacluster** user has full access to a cluster.

ha_cluster_manage_firewall: true
: A variable that determines whether the **ha_cluster** RHEL system role manages the firewall.

ha_cluster_manage_selinux: true
: A variable that determines whether the **ha_cluster** RHEL system role manages the ports of the firewall high availability service using the **selinux** RHEL system role.

ha_cluster_resource_primitives: *<cluster_resources>*
: A list of resource definitions for the Pacemaker resources configured by the ha_cluster RHEL system role, including fencing

ha_cluster_resource_groups: *<resource_groups>*
: A list of resource group definitions configured by the **ha_cluster** RHEL system role.

ha_cluster_resource_clones: *<resource_clones>*
: A list of resource clone definitions configured by the **ha_cluster** RHEL system role.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check --ask-vault-pass ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** file
- **/usr/share/doc/rhel-system-roles/ha_cluster/** directory
- Configuring fencing in a Red Hat High Availability cluster

11.6. CONFIGURING A HIGH AVAILABILITY CLUSTER WITH RESOURCE AND RESOURCE OPERATION DEFAULTS

(RHEL 8.9 and later) In your cluster configuration, you can change the Pacemaker default values of a resource option for all resources. You can also change the default value for all resource operations in the cluster.

For information about changing the default value of a resource option, see Changing the default value of a resource option. For information about global resource operation defaults, see Configuring global resource operation defaults.

The following example procedure uses the **ha_cluster** RHEL system role to create a high availability cluster that defines resource and resource operation defaults.

> **WARNING**
>
> The **ha_cluster** RHEL system role replaces any existing cluster configuration on the specified nodes. Any settings not specified in the playbook will be lost.

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 systems that you will use as your cluster members have active subscription coverage for RHEL and the RHEL High Availability Add-On.
- The inventory file specifies the cluster nodes as described in Specifying an inventory for the ha_cluster RHEL system role. For general information about creating an inventory file, see Preparing a control node on RHEL 8 .

Procedure

1. Store your sensitive variables in an encrypted file:
   a. Create the vault:

```
$ ansible-vault create ~/vault.yml
New Vault password: <vault_password>
Confirm New Vault password: <vault_password>
```

–

b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

```
cluster_password: <cluster_password>
```

c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

```
---
- name: Create a high availability cluster
  hosts: node1 node2
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Create cluster with fencing and resource operation defaults
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.ha_cluster
      vars:
        ha_cluster_cluster_name: my-new-cluster
        ha_cluster_hacluster_password: "{{ cluster_password }}"
        ha_cluster_manage_firewall: true
        ha_cluster_manage_selinux: true
        # Set a different resource-stickiness value during
        # and outside work hours. This allows resources to
        # automatically move back to their most
        # preferred hosts, but at a time that
        # does not interfere with business activities.
        ha_cluster_resource_defaults:
          meta_attrs:
            - id: core-hours
              rule: date-spec hours=9-16 weekdays=1-5
              score: 2
              attrs:
                - name: resource-stickiness
                  value: INFINITY
            - id: after-hours
              score: 1
              attrs:
                - name: resource-stickiness
                  value: 0
        # Default the timeout on all 10-second-interval
        # monitor actions on IPaddr2 resources to 8 seconds.
        ha_cluster_resource_operation_defaults:
          meta_attrs:
            - rule: resource ::IPaddr2 and op monitor interval=10s
              score: INFINITY
              attrs:
                - name: timeout
                  value: 8s
```

The settings specified in the example playbook include the following:

ha_cluster_cluster_name: *<cluster_name>*

The name of the cluster you are creating.

ha_cluster_hacluster_password: *<password>*

The password of the **hacluster** user. The **hacluster** user has full access to a cluster.

ha_cluster_manage_firewall: true

A variable that determines whether the **ha_cluster** RHEL system role manages the firewall.

ha_cluster_manage_selinux: true

A variable that determines whether the **ha_cluster** RHEL system role manages the ports of the firewall high availability service using the **selinux** RHEL system role.

ha_cluster_resource_defaults: *<resource_defaults>*

A variable that defines sets of resource defaults.

ha_cluster_resource_operation_defaults: *<resource_operation_defaults>*

A variable that defines sets of resource operation defaults.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check --ask-vault-pass ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** file
- **/usr/share/doc/rhel-system-roles/ha_cluster/** directory

11.7. CONFIGURING A HIGH AVAILABILITY CLUSTER WITH FENCING LEVELS

(RHEL 8.10 and later) When you configure multiple fencing devices for a node, you need to define fencing levels for those devices to determine the order that Pacemaker will use the devices to attempt to fence a node. For information about fencing levels, see Configuring fencing levels.

The following example procedure uses the **ha_cluster** RHEL system role to create a high availability cluster that defines fencing levels.

> **WARNING**
>
> The **ha_cluster** RHEL system role replaces any existing cluster configuration on the specified nodes. Any settings not specified in the playbook will be lost.

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 systems that you will use as your cluster members have active subscription coverage for RHEL and the RHEL High Availability Add-On.
- The inventory file specifies the cluster nodes as described in Specifying an inventory for the ha_cluster RHEL system role. For general information about creating an inventory file, see Preparing a control node on RHEL 8 .

Procedure

1. Store your sensitive variables in an encrypted file:
   a. Create the vault:

      ```
      $ ansible-vault create ~/vault.yml
      New Vault password: <vault_password>
      Confirm New Vault password: <vault_password>
      ```

   b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

      ```
      cluster_password: <cluster_password>
      fence1_password: <fence1_password>
      fence2_password: <fence2_password>
      ```

   c. Save the changes, and close the editor. Ansible encrypts the data in the vault.
2. Create a playbook file, for example **~/playbook.yml**. This example playbook file configures a cluster running the **firewalld** and **selinux** services.

   ```
   ---
   - name: Create a high availability cluster
     hosts: node1 node2
     vars_files:
       - ~/vault.yml
     tasks:
       - name: Configure a cluster that defines fencing levels
         ansible.builtin.include_role:
           name: redhat.rhel_system_roles.ha_cluster
   ```

```
vars:
  ha_cluster_cluster_name: my-new-cluster
  ha_cluster_hacluster_password: "{{ cluster_password }}"
  ha_cluster_manage_firewall: true
  ha_cluster_manage_selinux: true
  ha_cluster_resource_primitives:
    - id: apc1
      agent: 'stonith:fence_apc_snmp'
      instance_attrs:
        - attrs:
            - name: ip
              value: apc1.example.com
            - name: username
              value: user
            - name: password
              value: "{{ fence1_password }}"
            - name: pcmk_host_map
              value: node1:1;node2:2
    - id: apc2
      agent: 'stonith:fence_apc_snmp'
      instance_attrs:
        - attrs:
            - name: ip
              value: apc2.example.com
            - name: username
              value: user
            - name: password
              value: "{{ fence2_password }}"
            - name: pcmk_host_map
              value: node1:1;node2:2
  # Nodes have redundant power supplies, apc1 and apc2. Cluster must
  # ensure that when attempting to reboot a node, both power
  # supplies # are turned off before either power supply is turned
  # back on.
  ha_cluster_stonith_levels:
    - level: 1
      target: node1
      resource_ids:
        - apc1
        - apc2
    - level: 1
      target: node2
      resource_ids:
        - apc1
        - apc2
```

The settings specified in the example playbook include the following:

ha_cluster_cluster_name: *<cluster_name>*

The name of the cluster you are creating.

ha_cluster_hacluster_password: *<password>*

The password of the **hacluster** user. The **hacluster** user has full access to a cluster.

ha_cluster_manage_firewall: true

A variable that determines whether the **ha_cluster** RHEL system role manages the firewall.

ha_cluster_manage_selinux: true

A variable that determines whether the **ha_cluster** RHEL system role manages the ports of the firewall high availability service using the **selinux** RHEL system role.

ha_cluster_resource_primitives: *<cluster_resources>*

A list of resource definitions for the Pacemaker resources configured by the ha_cluster RHEL system role, including fencing

ha_cluster_stonith_levels: *<stonith_levels>*

A variable that defines STONITH levels, also known as fencing topology, which configure a cluster to use multiple devices to fence nodes.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check --ask-vault-pass ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** file
- **/usr/share/doc/rhel-system-roles/ha_cluster/** directory

11.8. CONFIGURING A HIGH AVAILABILITY CLUSTER WITH RESOURCE CONSTRAINTS

When configuring a cluster, you can specify the behavior of the cluster resources to be in line with your application requirements. You can control the behavior of cluster resources by configuring resource constraints.

You can define the following categories of resource constraints:

- Location constraints, which determine which nodes a resource can run on. For information about location constraints, see Determining which nodes a resource can run on .
- Ordering constraints, which determine the order in which the resources are run. For information about ordering constraints, see Determing the order in which cluster resources are run .
- Colocation constraints, which specify that the location of one resource depends on the location of another resource. For information about colocation constraints, see Colocating cluster resources.
- Ticket constraints, which indicate the resources that depend on a particular Booth ticket. For information about Booth ticket constraints, see Multi-site Pacemaker clusters.

The following example procedure uses the **ha_cluster** RHEL system role to create a high availability cluster that includes resource location constraints, resource colocation constraints, resource order constraints, and resource ticket constraints.

> **WARNING**
>
> The **ha_cluster** RHEL system role replaces any existing cluster configuration on the specified nodes. Any settings not specified in the playbook will be lost.

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 systems that you will use as your cluster members have active subscription coverage for RHEL and the RHEL High Availability Add-On.
- The inventory file specifies the cluster nodes as described in Specifying an inventory for the ha_cluster RHEL system role. For general information about creating an inventory file, see Preparing a control node on RHEL 8 .

Procedure

1. Store your sensitive variables in an encrypted file:
   a. Create the vault:

      ```
      $ ansible-vault create ~/vault.yml
      New Vault password: <vault_password>
      Confirm New Vault password: <vault_password>
      ```

   b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

      ```
      cluster_password: <cluster_password>
      ```

   c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

   ```
   ---
   - name: Create a high availability cluster
     hosts: node1 node2
     vars_files:
       - ~/vault.yml
     tasks:
       - name: Create cluster with resource constraints
         ansible.builtin.include_role:
   ```

```
  name: redhat.rhel_system_roles.ha_cluster
 vars:
  ha_cluster_cluster_name: my-new-cluster
  ha_cluster_hacluster_password: "{{ cluster_password }}"
  ha_cluster_manage_firewall: true
  ha_cluster_manage_selinux: true
  # 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
```

The settings specified in the example playbook include the following:

ha_cluster_cluster_name: *<cluster_name>*
: The name of the cluster you are creating.

ha_cluster_hacluster_password: *<password>*
: The password of the **hacluster** user. The **hacluster** user has full access to a cluster.

ha_cluster_manage_firewall: true
: A variable that determines whether the **ha_cluster** RHEL system role manages the firewall.

ha_cluster_manage_selinux: true
: A variable that determines whether the **ha_cluster** RHEL system role manages the ports of the firewall high availability service using the **selinux** RHEL system role.

ha_cluster_resource_primitives: *<cluster_resources>*
: A list of resource definitions for the Pacemaker resources configured by the ha_cluster RHEL system role, including fencing

ha_cluster_constraints_location: *<location_constraints>*
: A variable that defines resource location constraints.

ha_cluster_constraints_colocation: *<colocation_constraints>*
: A variable that defines resource colocation constraints.

ha_cluster_constraints_order: *<order_constraints>*
: A variable that defines resource order constraints.

ha_cluster_constraints_ticket: *<ticket_constraints>*
: A variable that defines Booth ticket constraints.

3. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check --ask-vault-pass ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Additional resources

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

- **/usr/share/doc/rhel-system-roles/ha_cluster/** directory

11.9. CONFIGURING COROSYNC VALUES IN A HIGH AVAILABILITY CLUSTER

(RHEL 8.7 and later) The **corosync.conf** file provides the cluster parameters used by Corosync, the cluster membership and messaging layer that Pacemaker is built on. For your system configuration, you can change some of the default parameters in the **corosync.conf** file. In general, you should not edit the **corosync.conf** file directly. You can, however, configure Corosync values by using the **ha_cluster** RHEL system role.

The following example procedure uses the **ha_cluster** RHEL system role to create a high availability cluster that configures Corosync values.

> **WARNING**
>
> The **ha_cluster** RHEL system role replaces any existing cluster configuration on the specified nodes. Any settings not specified in the playbook will be lost.

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 systems that you will use as your cluster members have active subscription coverage for RHEL and the RHEL High Availability Add-On.
- The inventory file specifies the cluster nodes as described in Specifying an inventory for the ha_cluster RHEL system role. For general information about creating an inventory file, see Preparing a control node on RHEL 8 .

Procedure

1. Store your sensitive variables in an encrypted file:
    a. Create the vault:

    ```
    $ ansible-vault create ~/vault.yml
    New Vault password: <vault_password>
    Confirm New Vault password: <vault_password>
    ```

    b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

    ```
    cluster_password: <cluster_password>
    ```

    c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

```
---
- name: Create a high availability cluster
  hosts: node1 node2
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Create cluster that configures Corosync values
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.ha_cluster
      vars:
        ha_cluster_cluster_name: my-new-cluster
        ha_cluster_hacluster_password: "{{ cluster_password }}"
        ha_cluster_manage_firewall: true
        ha_cluster_manage_selinux: true
        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
```

The settings specified in the example playbook include the following:

ha_cluster_cluster_name: *<cluster_name>*
: The name of the cluster you are creating.

ha_cluster_hacluster_password: *<password>*
: The password of the **hacluster** user. The **hacluster** user has full access to a cluster.

ha_cluster_manage_firewall: true
: A variable that determines whether the **ha_cluster** RHEL system role manages the firewall.

ha_cluster_manage_selinux: true
: A variable that determines whether the **ha_cluster** RHEL system role manages the ports of the firewall high availability service using the **selinux** RHEL system role.

ha_cluster_transport: *<transport_method>*
: A variable that sets the cluster transport method.

ha_cluster_totem: *<totem_options>*
: A variable that configures Corosync totem options.

ha_cluster_quorum: *<quorum_options>*
: A variable that configures cluster quorum options.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --ask-vault-pass --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** file
- **/usr/share/doc/rhel-system-roles/ha_cluster/** directory

11.10. CONFIGURING A HIGH AVAILABILITY CLUSTER WITH SBD NODE FENCING

(RHEL 8.7 and later) The following procedure uses the **ha_cluster** RHEL system role to create a high availability cluster that uses SBD node fencing.

> **WARNING**
>
> The **ha_cluster** RHEL system role replaces any existing cluster configuration on the specified nodes. Any settings not specified in the playbook will be lost.

This playbook uses an inventory file that loads a watchdog module (supported in RHEL 8.9 and later) as described in Configuring watchdog and SBD devices in an inventory .

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 systems that you will use as your cluster members have active subscription coverage for RHEL and the RHEL High Availability Add-On.
- The inventory file specifies the cluster nodes as described in Specifying an inventory for the ha_cluster RHEL system role.

Procedure

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

```
---
- name: Create a high availability cluster that uses SBD node fencing
  hosts: node1 node2
  roles:
    - rhel-system-roles.ha_cluster
  vars:
    ha_cluster_cluster_name: my-new-cluster
    ha_cluster_hacluster_password: <password>
    ha_cluster_manage_firewall: true
    ha_cluster_manage_selinux: true
    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: 30
    # Suggested optimal values for SBD timeouts:
    # watchdog-timeout * 2 = msgwait-timeout (set automatically)
    # msgwait-timeout * 1.2 = stonith-timeout
    ha_cluster_cluster_properties:
      - attrs:
```

```
      - name: stonith-timeout
        value: 72
  ha_cluster_resource_primitives:
    - id: fence_sbd
      agent: 'stonith:fence_sbd'
      instance_attrs:
        - attrs:
            # taken from host_vars
            - name: devices
              value: "{{ ha_cluster.sbd_devices | join(',') }}"
            - name: pcmk_delay_base
              value: 30
```

This example playbook file configures a cluster running the **firewalld** and **selinux** services that uses SBD fencing and creates the SBD Stonith resource.

When creating your playbook file for production, vault encrypt the password, as described in Encrypting content with Ansible Vault .

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** file
- **/usr/share/doc/rhel-system-roles/ha_cluster/** directory

11.11. CONFIGURING A HIGH AVAILABILITY CLUSTER USING A QUORUM DEVICE

(RHEL 8.8 and later) Your cluster can sustain more node failures than standard quorum rules permit when you configure a separate quorum device. The quorum device acts as a lightweight arbitration device for the cluster. A quorum device is recommended for clusters with an even number of nodes. With two-node clusters, the use of a quorum device can better determine which node survives in a split-brain situation.

For information about quorum devices, see Configuring quorum devices.

To configure a high availability cluster with a separate quorum device by using the **ha_cluster** RHEL system role, first set up the quorum device. After setting up the quorum device, you can use the device in any number of clusters.

11.11.1. Configuring a quorum device

To configure a quorum device using the **ha_cluster** RHEL system role, follow the steps in this example procedure. Note that you cannot run a quorum device on a cluster node.

> **WARNING**
>
> The **ha_cluster** RHEL system role replaces any existing cluster configuration on the specified nodes. Any settings not specified in the playbook will be lost.

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 system that you will use to run the quorum device has active subscription coverage for RHEL and the RHEL High Availability Add-On.
- The inventory file specifies the quorum devices as described in Specifying an inventory for the ha_cluster RHEL system role. For general information about creating an inventory file, see Preparing a control node on RHEL 8 .

Procedure

1. Store your sensitive variables in an encrypted file:
   a. Create the vault:

      ```
      $ ansible-vault create ~/vault.yml
      New Vault password: <vault_password>
      Confirm New Vault password: <vault_password>
      ```

   b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

      ```
      cluster_password: <cluster_password>
      ```

   c. Save the changes, and close the editor. Ansible encrypts the data in the vault.
2. Create a playbook file, for example ~/**playbook-qdevice.yml**, with the following content:

   ```
   ---
   - name: Configure a host with a quorum device
     hosts: nodeQ
     vars_files:
       - ~/vault.yml
     tasks:
       - name: Create a quorum device for the cluster
         ansible.builtin.include_role:
           name: redhat.rhel_system_roles.ha_cluster
   ```

```
vars:
  ha_cluster_cluster_present: false
  ha_cluster_hacluster_password: "{{ cluster_password }}"
  ha_cluster_manage_firewall: true
  ha_cluster_manage_selinux: true
  ha_cluster_qnetd:
    present: true
```

The settings specified in the example playbook include the following:

ha_cluster_cluster_present: false
: A variable that, if set to **false**, determines that all cluster configuration will be removed from the target host.

ha_cluster_hacluster_password: *<password>*
: The password of the **hacluster** user. The **hacluster** user has full access to a cluster.

ha_cluster_manage_firewall: true
: A variable that determines whether the **ha_cluster** RHEL system role manages the firewall.

ha_cluster_manage_selinux: true
: A variable that determines whether the **ha_cluster** RHEL system role manages the ports of the firewall high availability service using the **selinux** RHEL system role.

ha_cluster_qnetd: *<quorum_device_options>*
: A variable that configures a **qnetd** host.

3. Validate the playbook syntax:

```
$ ansible-playbook --ask-vault-pass --syntax-check ~/playbook-qdevice.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook-qdevice.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** file
- **/usr/share/doc/rhel-system-roles/ha_cluster/** directory

11.11.2. Configuring a cluster to use a quorum device

To configure a cluster to use a quorum device, follow the steps in this example procedure.

> **WARNING**
>
> The **ha_cluster** RHEL system role replaces any existing cluster configuration on the specified nodes. Any settings not specified in the playbook will be lost.

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 systems that you will use as your cluster members have active subscription coverage for RHEL and the RHEL High Availability Add-On.
- The inventory file specifies the cluster nodes as described in Specifying an inventory for the ha_cluster RHEL system role. For general information about creating an inventory file, see Preparing a control node on RHEL 8 .
- You have configured a quorum device.

Procedure

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

```
---
- name: Configure a cluster to use a quorum device
  hosts: node1 node2
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Create cluster that uses a quorum device
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.ha_cluster
      vars:
        ha_cluster_cluster_name: my-new-cluster
        ha_cluster_hacluster_password: "{{ cluster_password }}"
        ha_cluster_manage_firewall: true
        ha_cluster_manage_selinux: true
        ha_cluster_quorum:
          device:
            model: net
            model_options:
              - name: host
                value: nodeQ
              - name: algorithm
                value: lms
```

The settings specified in the example playbook include the following:

ha_cluster_cluster_name: *<cluster_name>*

The name of the cluster you are creating.

ha_cluster_hacluster_password: *<password>*

The password of the **hacluster** user. The **hacluster** user has full access to a cluster.

ha_cluster_manage_firewall: true

A variable that determines whether the **ha_cluster** RHEL system role manages the firewall.

ha_cluster_manage_selinux: true

A variable that determines whether the **ha_cluster** RHEL system role manages the ports of the firewall high availability service using the **selinux** RHEL system role.

ha_cluster_quorum: *<quorum_parameters>*

A variable that configures cluster quorum which you can use to specify that the cluster uses a quorum device.

2. Validate the playbook syntax:

```
$ ansible-playbook --ask-vault-pass --syntax-check ~/playbook-cluster-qdevice.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook-cluster-qdevice.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** file
- **/usr/share/doc/rhel-system-roles/ha_cluster/** directory

11.12. CONFIGURING A HIGH AVAILABILITY CLUSTER WITH NODE ATTRIBUTES

(RHEL 8.10 and later) You can use Pacemaker rules to make your configuration more dynamic. For example, you can use a node attribute to assign machines to different processing groups based on time and then use that attribute when creating location constraints.

Node attribute expressions are used to control a resource based on the attributes defined by a node or nodes. For information on node attributes, see Determining resource location with rules.

The following example procedure uses the **ha_cluster** RHEL system role to create a high availability cluster that configures node attributes.

> **WARNING**
>
> The **ha_cluster** RHEL system role replaces any existing cluster configuration on the specified nodes. Any settings not specified in the playbook will be lost.

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 systems that you will use as your cluster members have active subscription coverage for RHEL and the RHEL High Availability Add-On.
- The inventory file specifies the cluster nodes as described in Specifying an inventory for the ha_cluster RHEL system role. For general information about creating an inventory file, see Preparing a control node on RHEL 8 .

Procedure

1. Store your sensitive variables in an encrypted file:
   a. Create the vault:

      ```
      $ ansible-vault create ~/vault.yml
      New Vault password: <vault_password>
      Confirm New Vault password: <vault_password>
      ```

   b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

      ```
      cluster_password: <cluster_password>
      ```

   c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

```
---
- name: Create a high availability cluster
  hosts: node1 node2
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Create a cluster that defines node attributes
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.ha_cluster
      vars:
        ha_cluster_cluster_name: my-new-cluster
        ha_cluster_hacluster_password: "{{ cluster_password }}"
        ha_cluster_manage_firewall: true
        ha_cluster_manage_selinux: true
        ha_cluster_node_options:
          - node_name: node1
            attributes:
              - attrs:
                  - name: attribute1
                    value: value1A
                  - name: attribute2
                    value: value2A
```

```
    - node_name: node2
      attributes:
        - attrs:
            - name: attribute1
              value: value1B
            - name: attribute2
              value: value2B
```

ha_cluster_cluster_name: *<cluster_name>*
: The name of the cluster you are creating.

ha_cluster_hacluster_password: *<password>*
: The password of the **hacluster** user. The **hacluster** user has full access to a cluster.

ha_cluster_manage_firewall: true
: A variable that determines whether the **ha_cluster** RHEL system role manages the firewall.

ha_cluster_manage_selinux: true
: A variable that determines whether the **ha_cluster** RHEL system role manages the ports of the firewall high availability service using the **selinux** RHEL system role.

ha_cluster_node_options: *<node_settings>*
: A variable that defines various settings that vary from one cluster node to another.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check --ask-vault-pass ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Additional resources

- Pacemaker rules

11.13. CONFIGURING AN APACHE HTTP SERVER IN A HIGH AVAILABILITY CLUSTER WITH THE HA_CLUSTER RHEL SYSTEM ROLE

High availability clusters provide highly available services by eliminating single points of failure and by failing over services from one cluster node to another in case a node becomes inoperative. Red Hat provides a variety of documentation for planning, configuring, and maintaining a Red Hat high availability cluster. For a listing of articles that provide indexes to the various areas of Red Hat cluster documentation, see the Red Hat High Availability Add-On Documentation Guide .

The following example use case configures an active/passive Apache HTTP server in a two-node Red Hat Enterprise Linux High Availability Add-On cluster by using the **ha_cluster** RHEL system role. In this use case, clients access the Apache HTTP server through a floating IP address. The web server runs on one of two nodes in the cluster. If the node on which the web server is running becomes inoperative, the web server starts up again on the second node of the cluster with minimal service interruption.

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.

> **WARNING**
>
> The **ha_cluster** RHEL system role replaces any existing cluster configuration on the specified nodes. Any settings not specified in the playbook will be lost.

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 systems that you will use as your cluster members have active subscription coverage for RHEL and the RHEL High Availability Add-On.
- The inventory file specifies the cluster nodes as described in Specifying an inventory for the ha_cluster RHEL system role. For general information about creating an inventory file, see Preparing a control node on RHEL 8 .
- You have configured an LVM logical volume with an XFS file system, as described in Configuring an LVM volume with an XFS 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.

Procedure

1. Store your sensitive variables in an encrypted file:

    a. Create the vault:

    ```
    $ ansible-vault create ~/vault.yml
    New Vault password: <vault_password>
    Confirm New Vault password: <vault_password>
    ```

    b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

    ```
    cluster_password: <cluster_password>
    ```

    c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

```
---
- name: Create a high availability cluster
  hosts: z1.example.com z2.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Configure active/passive Apache server in a high availability cluster
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.ha_cluster
      vars:
        ha_cluster_hacluster_password: "{{ cluster_password }}"
        ha_cluster_cluster_name: my_cluster
        ha_cluster_manage_firewall: true
        ha_cluster_manage_selinux: true
        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: xfs
          - 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
```

The settings specified in the example playbook include the following:

ha_cluster_cluster_name: *<cluster_name>*
: The name of the cluster you are creating.

ha_cluster_hacluster_password: *<password>*
: The password of the **hacluster** user. The **hacluster** user has full access to a cluster.

ha_cluster_manage_firewall: true
: A variable that determines whether the **ha_cluster** RHEL system role manages the firewall.

ha_cluster_manage_selinux: true
: A variable that determines whether the **ha_cluster** RHEL system role manages the ports of the firewall high availability service using the **selinux** RHEL system role.

ha_cluster_fence_agent_packages: *<fence_agent_packages>*
: A list of fence agent packages to install.

ha_cluster_resource_primitives: *<cluster_resources>*
: A list of resource definitions for the Pacemaker resources configured by the ha_cluster RHEL system role, including fencing

ha_cluster_resource_groups: *<resource_groups>*
: A list of resource group definitions configured by the **ha_cluster** RHEL system role.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check --ask-vault-pass ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

5. When you use the **apache** resource agent to manage Apache, it does not use **systemd**. Because of this, you must edit the **logrotate** script supplied with Apache so that it does not use **systemctl** to reload Apache.
   Remove the following line in the **/etc/logrotate.d/httpd** file on each node in the cluster.

   ```
   # /bin/systemctl reload httpd.service > /dev/null 2>/dev/null || true
   ```

   - For RHEL 8.6 and later, replace the line you removed with the following three lines, specifying **/var/run/httpd-*website*.pid** as the PID file path where *website* is the name of the Apache resource. In this example, the Apache resource name is **Website**.

     ```
     /usr/bin/test -f /var/run/httpd-Website.pid >/dev/null 2>/dev/null &&
     /usr/bin/ps -q $(/usr/bin/cat /var/run/httpd-Website.pid) >/dev/null 2>/dev/null &&
     /usr/sbin/httpd -f /etc/httpd/conf/httpd.conf -c "PidFile /var/run/httpd-Website.pid" -k
     graceful > /dev/null 2>/dev/null || true
     ```

   - For RHEL 8.5 and earlier, replace the line you removed with the following three lines.

     ```
     /usr/bin/test -f /run/httpd.pid >/dev/null 2>/dev/null &&
     /usr/bin/ps -q $(/usr/bin/cat /run/httpd.pid) >/dev/null 2>/dev/null &&
     /usr/sbin/httpd -f /etc/httpd/conf/httpd.conf -c "PidFile /run/httpd.pid" -k graceful > /dev/null
     2>/dev/null || true
     ```

Verification

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 about the **resource-stickiness** meta attribute, see Configuring a resource to prefer its current node .

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** file
- **/usr/share/doc/rhel-system-roles/ha_cluster/** directory

CHAPTER 12. CONFIGURING THE SYSTEMD JOURNAL BY USING RHEL SYSTEM ROLES

With the **journald** RHEL system role you can automate the **systemd** journal, and configure persistent logging by using the Red Hat Ansible Automation Platform.

12.1. CONFIGURING PERSISTENT LOGGING BY USING THE JOURNALD RHEL SYSTEM ROLE

By default, the **systemd** journal stores logs only in a small ring buffer in **/run/log/journal**, which is not persistent. Rebooting the system also removes journal database logs. You can configure persistent logging consistently on multiple systems by using the **journald** RHEL system role.

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.

Procedure

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

```
---
- name: Configure journald
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure persistent logging
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.journald
      vars:
        journald_persistent: true
        journald_max_disk_size: <size>
        journald_per_user: true
        journald_sync_interval: <interval>
```

The settings specified in the example playbook include the following:

journald_persistent: true
: Enables persistent logging.

journald_max_disk_size: *<size>*
: Specifies the maximum size of disk space for journal files in MB, for example, **2048**.

journald_per_user: true
: Configures **journald** to keep log data separate for each user.

journald_sync_interval: *<interval>*
: Sets the synchronization interval in minutes, for example, **1**.
For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.journald/README.md** file on the control node.

2. Validate the playbook syntax:

   ```
   $ ansible-playbook --syntax-check ~/playbook.yml
   ```

   Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

   ```
   $ ansible-playbook ~/playbook.yml
   ```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.journald/README.md** file
- **/usr/share/doc/rhel-system-roles/journald/** directory

CHAPTER 13. CONFIGURING AUTOMATIC CRASH DUMPS BY 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.

13.1. CONFIGURING THE KERNEL CRASH DUMPING MECHANISM BY USING THE KDUMP RHEL SYSTEM ROLE

Kernel crash dumping is a crucial feature for diagnosing and troubleshooting system issues. When your system encounters a kernel panic or other critical failure, crash kernel dumping allows you to capture a memory dump (core dump) of the kernel's state at the time of the failure.

By using an Ansible playbook, you can set kernel crash dump parameters on multiple systems using the **kdump** RHEL system role. This ensures consistent settings across all managed nodes for the **kdump** service.

> **WARNING**
>
> The **kdump** system role replaces the content in the **/etc/kdump.conf** and **/etc/sysconfig/kdump** configuration files. Previous settings are changed to those specified in the role variables, and lost if they are not specified in the role variables.

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.

Procedure

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

```
---
- name: Configuring kernel crash dumping
  hosts: managed-node-01.example.com
  tasks:
    - name: Setting the kdump directory.
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.kdump
      vars:
        kdump_target:
          type: raw
```

```
      location: /dev/sda1
    kdump_path: /var/crash/vmcore
    kernel_settings_reboot_ok: true
```

The settings specified in the example playbook include the following:

kdump_target: *<type_and_location>*
: Writes **vmcore** to a location other than the root file system. The **location** refers to a partition (by name, label, or UUID) when the **type** is raw or file system.

kernel_settings_reboot_ok: *<true|false>*
: The default is **false**. If set to **true**, the system role will determine if a reboot of the managed host is necessary for the requested changes to take effect and reboot it. If set to **false**, the role will return the variable **kernel_settings_reboot_required** with a value of **true**, indicating that a reboot is required. In this case, a user must reboot the managed node manually.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.kdump/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Verify the kernel crash dump parameters:

```
$ ansible managed-node-01.example.com -m command -a 'grep crashkernel /proc/cmdline'
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.kdump/README.md** file
- **/usr/share/doc/rhel-system-roles/kdump/** directory

CHAPTER 14. CONFIGURING KERNEL PARAMETERS PERMANENTLY BY USING RHEL SYSTEM ROLES

You can use the **kernel_settings** RHEL system role to configure kernel parameters on multiple clients simultaneously. Simultaneous configuration has the following advantages:

- 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.

14.1. APPLYING SELECTED KERNEL PARAMETERS BY USING THE KERNEL_SETTINGS RHEL SYSTEM ROLE

You can use the **kernel_settings** RHEL system role to remotely configure various kernel parameters across multiple managed operating systems with persistent effects. For example, you can configure:

- Transparent hugepages to increase performance by reducing the overhead of managing smaller pages.
- The largest packet sizes to be transmitted over the network with the loopback interface.
- Limits on files to be opened simultaneously.

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.

Procedure

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

```
---
- name: Configuring kernel settings
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure hugepages, packet size for loopback device, and limits on
simultaneously open files.
      ansible.builtin.include_role:
        name: redhat.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
    kernel_settings_reboot_ok: true
```

The settings specified in the example playbook include the following:

kernel_settings_sysfs: *<list_of_sysctl_settings>*
: A YAML list of **sysctl** settings and the values you want to assign to these settings.

kernel_settings_transparent_hugepages: *<value>*
: Controls the memory subsystem Transparent Huge Pages (THP) setting. You can disable THP support (**never**), enable it system wide (**always**) or inside **MAD_HUGEPAGE** regions (**madvise**).

kernel_settings_reboot_ok: *<true|false>*
: The default is **false**. If set to **true**, the system role will determine if a reboot of the managed host is necessary for the requested changes to take effect and reboot it. If set to **false**, the role will return the variable **kernel_settings_reboot_required** with a value of **true**, indicating that a reboot is required. In this case, a user must reboot the managed node manually.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.kdump/README.md** file on the control node.

1. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

2. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Verify the affected kernel parameters:

```
# ansible managed-node-01.example.com -m command -a 'sysctl fs.file-max kernel.threads-max net.ipv6.conf.lo.mtu'
# ansible managed-node-01.example.com -m command -a 'cat /sys/kernel/mm/transparent_hugepage/enabled'
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.kernel_settings/README.md** file
- **/usr/share/doc/rhel-system-roles/kernel_settings/** directory

CHAPTER 15. CONFIGURING LOGGING BY USING RHEL SYSTEM ROLES

You can use the **logging** RHEL system role to configure your local and remote hosts as logging servers in an automated fashion to collect logs from many client systems.

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 engine
- Logs forwarded to another logging system

With the **logging** RHEL 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.

15.1. FILTERING LOCAL LOG MESSAGES BY USING THE LOGGING RHEL SYSTEM ROLE

You can use the property-based filter of the **logging** RHEL system role to filter your local log messages based on various conditions. As a result, you can achieve for example:

- Log clarity: In a high-traffic environment, logs can grow rapidly. The focus on specific messages, like errors, can help to identify problems faster.
- Optimized system performance: Excessive amount of logs is usually connected with system performance degradation. Selective logging for only the important events can prevent resource depletion, which enables your systems to run more efficiently.
- Enhanced security: Efficient filtering through security messages, like system errors and failed logins, helps to capture only the relevant logs. This is important for detecting breaches and meeting compliance standards.

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.

Procedure

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

```
---
- name: Deploy the logging solution
  hosts: managed-node-01.example.com
  tasks:
    - name: Filter logs based on a specific value they contain
      ansible.builtin.include_role:
        name: redhat.rhel_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]
```

The settings specified in the example playbook include the following:

logging_inputs
: Defines a list of logging input dictionaries. The **type: basics** option covers inputs from **systemd** journal or Unix socket.

logging_outputs
: Defines a list of logging output dictionaries. The **type: files** option supports storing logs in the local files, usually in the **/var/log/** directory. The **property: msg**; **property: contains**; and **property_value: error** options specify that all logs that contain the **error** string are stored in the **/var/log/errors.log** file. The **property: msg**; **property: !contains**; and **property_value: error** options specify that all other logs are put in the **/var/log/others.log** file. You can replace the **error** value with the string by which you want to filter.

logging_flows
: Defines a list of logging flow dictionaries to specify relationships between **logging_inputs** and **logging_outputs**. The **inputs: [files_input]** option specifies a list of inputs, from which processing of logs starts. The **outputs: [files_output0, files_output1]** option specifies a list of outputs, to which the logs are sent.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

1. On the managed node, test the syntax of the **/etc/rsyslog.conf** file:

```
# rsyslogd -N 1
rsyslogd: version 8.1911.0-6.el8, config validation run...
rsyslogd: End of config validation run. Bye.
```

2. On the managed node, 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

- **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** file
- **/usr/share/doc/rhel-system-roles/logging/** directory
- **rsyslog.conf(5)** and **syslog(3)** man pages on your system

15.2. APPLYING A REMOTE LOGGING SOLUTION BY USING THE LOGGING RHEL SYSTEM ROLE

You can use the **logging** RHEL system role to configure a remote logging solution, where one or more clients take logs from the **systemd-journal** service and forward them to a remote server. The server receives remote input from the **remote_rsyslog** and **remote_files** configurations, and outputs the logs to local files in directories named by remote host names.

As a result, you can cover use cases where you need for example:

- Centralized log management: Collecting, accessing, and managing log messages of multiple machines from a single storage point simplifies day-to-day monitoring and troubleshooting tasks. Also, this use case reduces the need to log into individual machines to check the log messages.

- Enhanced security: Storing log messages in one central place increases chances they are in a secure and tamper-proof environment. Such an environment makes it easier to detect and respond to security incidents more effectively and to meet audit requirements.
- Improved efficiency in log analysis: Correlating log messages from multiple systems is important for fast troubleshooting of complex problems that span multiple machines or services. That way you can quickly analyze and cross-reference events from different sources.

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.
- Define the ports in the SELinux policy of the server or client system and open the firewall for those ports. The default SELinux policy includes ports 601, 514, 6514, 10514, and 20514. To use a different port, see modify the SELinux policy on the client and server systems .

Procedure

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

```
---
- name: Deploy the logging solution
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure the server to receive remote input
      ansible.builtin.include_role:
        name: redhat.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: Deploy the logging solution
  hosts: managed-node-02.example.com
  tasks:
    - name: Configure the server to output the logs to local files in directories named by
remote host names
      ansible.builtin.include_role:
        name: redhat.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]
```

The settings specified in the first play of the example playbook include the following:

logging_inputs

Defines a list of logging input dictionaries. The **type: remote** option covers remote inputs from the other logging system over the network. The **udp_ports: [601]** option defines a list of UDP port numbers to monitor. The **tcp_ports: [601]** option defines a list of TCP port numbers to monitor. If both **udp_ports** and **tcp_ports** is set, **udp_ports** is used and **tcp_ports** is dropped.

logging_outputs

Defines a list of logging output dictionaries. The **type: remote_files** option makes output store logs to the local files per remote host and program name originated the logs.

logging_flows

Defines a list of logging flow dictionaries to specify relationships between **logging_inputs** and **logging_outputs**. The **inputs: [remote_udp_input, remote_tcp_input]** option specifies a list of inputs, from which processing of logs starts. The **outputs: [remote_files_output]** option specifies a list of outputs, to which the logs are sent.

The settings specified in the second play of the example playbook include the following:

logging_inputs

Defines a list of logging input dictionaries. The **type: basics** option covers inputs from **systemd** journal or Unix socket.

logging_outputs

Defines a list of logging output dictionaries. The **type: forwards** option supports sending logs to the remote logging server over the network. The **severity: info** option refers to log messages of the informative importance. The **facility: mail** option refers to the type of system program that is generating the log message. The **target: *<host1.example.com>*** option specifies the hostname of the remote logging server. The **udp_port: 601**/**tcp_port: 601** options define the UDP/TCP ports on which the remote logging server listens.

logging_flows

Defines a list of logging flow dictionaries to specify relationships between **logging_inputs** and **logging_outputs**. The **inputs: [basic_input]** option specifies a list of inputs, from which processing of logs starts. The **outputs: [forward_output0, forward_output1]** option specifies a list of outputs, to which the logs are sent.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

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/*<host2.example.com>*/messages** log, for example:

   ```
   # cat /var/log/<host2.example.com>/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

- **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** file
- **/usr/share/doc/rhel-system-roles/logging/** directory
- **rsyslog.conf(5)** and **syslog(3)** manual pages

15.3. USING THE LOGGING RHEL SYSTEM ROLE WITH TLS

Transport Layer Security (TLS) is a cryptographic protocol designed to allow secure communication over the computer network.

You can use the **logging** RHEL system role to configure a secure transfer of log messages, where one or more clients take logs from the **systemd-journal** service and transfer them to a remote server while using TLS.

Typically, TLS for transferring logs in a remote logging solution is used when sending sensitive data over less trusted or public networks, such as the Internet. Also, by using certificates in TLS you can ensure that the client is forwarding logs to the correct and trusted server. This prevents attacks like "man-in-the-middle".

15.3.1. Configuring client logging with TLS

You can use the **logging** RHEL system role to configure logging on RHEL clients and transfer logs to a remote logging system using TLS encryption.

This procedure creates a private key and a certificate. Next, it 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.

> **NOTE**
>
> You do not have to call the **certificate** RHEL system role in the playbook to create the certificate. The **logging** RHEL system role calls it automatically when the **logging_certificates** variable is set.
>
> In order for the CA to be able to sign the created certificate, the managed nodes must be enrolled in an IdM domain.

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 managed nodes are enrolled in an IdM domain.

Procedure

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

```
---
- name: Configure remote logging solution using TLS for secure transfer of logs
  hosts: managed-node-01.example.com
  tasks:
    - name: Deploying files input and forwards output with certs
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.logging
      vars:
        logging_certificates:
          - name: logging_cert
            dns: ['www.example.com']
            ca: ipa
            principal: "logging/{{ inventory_hostname }}@IDM.EXAMPLE.COM"
        logging_pki_files:
```

```
      - ca_cert: /local/path/to/ca_cert.pem
        cert: /local/path/to/logging_cert.pem
        private_key: /local/path/to/logging_cert.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 settings specified in the example playbook include the following:

logging_certificates

The value of this parameter is passed on to **certificate_requests** in the **certificate** RHEL system role and used to create a private key and certificate.

logging_pki_files

Using this parameter, you can configure the paths and other settings that logging uses to find the CA, certificate, and key files used for TLS, specified with one or more of the following sub-parameters: **ca_cert**, **ca_cert_src**, **cert**, **cert_src**, **private_key**, **private_key_src**, and **tls**.

> NOTE
>
> If you are using **logging_certificates** to create the files on the managed node, do not use **ca_cert_src**, **cert_src**, and **private_key_src**, which are used to copy files not created by **logging_certificates**.

ca_cert

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

cert

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

ca_cert_src

Represents the path to the CA certificate file on the control node which is copied to the target host to the location specified by **ca_cert**. Do not use this if using **logging_certificates**.

cert_src

Represents the path to a certificate file on the control node which is copied to the target host to the location specified by **cert**. Do not use this if using **logging_certificates**.

private_key_src

Represents the path to a private key file on the control node which is copied to the target host to the location specified by **private_key**. Do not use this if using **logging_certificates**.

tls

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

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** file
- **/usr/share/doc/rhel-system-roles/logging/** directory
- **/usr/share/ansible/roles/rhel-system-roles.certificate/README.md** file
- **/usr/share/doc/rhel-system-roles/certificate/** directory
- Requesting certificates using RHEL system roles .
- **rsyslog.conf(5)** and **syslog(3)** manual pages

15.3.2. Configuring server logging with TLS

You can use the **logging** RHEL system role to configure logging on RHEL servers and set them to receive logs from a remote logging system using TLS encryption.

This procedure creates a private key and a certificate. Next, it configures TLS on all hosts in the server group in the Ansible inventory.

NOTE

You do not have to call the **certificate** RHEL system role in the playbook to create the certificate. The **logging** RHEL system role calls it automatically.

In order for the CA to be able to sign the created certificate, the managed nodes must be enrolled in an IdM domain.

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 managed nodes are enrolled in an IdM domain.

Procedure

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

```
---
- name: Configure remote logging solution using TLS for secure transfer of logs
  hosts: managed-node-01.example.com
  tasks:
    - name: Deploying remote input and remote_files output with certs
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.logging
      vars:
        logging_certificates:
          - name: logging_cert
            dns: ['www.example.com']
            ca: ipa
            principal: "logging/{{ inventory_hostname }}@IDM.EXAMPLE.COM"
        logging_pki_files:
          - ca_cert: /local/path/to/ca_cert.pem
            cert: /local/path/to/logging_cert.pem
            private_key: /local/path/to/logging_cert.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 settings specified in the example playbook include the following:

logging_certificates

The value of this parameter is passed on to **certificate_requests** in the **certificate** RHEL system role and used to create a private key and certificate.

logging_pki_files

Using this parameter, you can configure the paths and other settings that logging uses to find the CA, certificate, and key files used for TLS, specified with one or more of the following sub-parameters: **ca_cert**, **ca_cert_src**, **cert**, **cert_src**, **private_key**, **private_key_src**, and **tls**.

NOTE

If you are using **logging_certificates** to create the files on the managed node, do not use **ca_cert_src**, **cert_src**, and **private_key_src**, which are used to copy files not created by **logging_certificates**.

ca_cert

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

cert

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

ca_cert_src

Represents the path to the CA certificate file on the control node which is copied to the target host to the location specified by **ca_cert**. Do not use this if using **logging_certificates**.

cert_src

Represents the path to a certificate file on the control node which is copied to the target host to the location specified by **cert**. Do not use this if using **logging_certificates**.

private_key_src

Represents the path to a private key file on the control node which is copied to the target host to the location specified by **private_key**. Do not use this if using **logging_certificates**.

tls

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

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** file
- **/usr/share/doc/rhel-system-roles/logging/** directory
- Requesting certificates using RHEL system roles .
- **rsyslog.conf(5)** and **syslog(3)** manual pages

15.4. USING THE LOGGING RHEL 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 the 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. To achieve that use case, you can use the **logging** RHEL system role to configure the logging system to reliably send and receive log entries.

15.4.1. Configuring client logging with RELP

You can use the **logging** RHEL system role to configure a transfer of log messages stored locally to the remote logging system with RELP.

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 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.

Procedure

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

```
---
- name: Configure client-side of the remote logging solution using RELP
  hosts: managed-node-01.example.com
  tasks:
    - name: Deploy basic input and RELP output
      ansible.builtin.include_role:
        name: redhat.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 settings specified in the example playbook include the following:

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 the {**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 the {**ca_cert**, **cert**, **private_key**} triplet is set, files are expected to be on the default path before the logging configuration.
- If both 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 certificate. 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 certificate file path which is copied to the managed node. If **ca_cert** is specified, it is copied to the location.

cert_src

Represents the local certificate file path which is copied to the managed node. If **cert** is specified, it is copied to the location.

private_key_src

Represents the local key file path which is copied to the managed node. 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.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** file
- **/usr/share/doc/rhel-system-roles/logging/** directory
- **rsyslog.conf(5)** and **syslog(3)** manual pages

15.4.2. Configuring server logging with RELP

You can use the **logging** RHEL system role to configure a server for receiving log messages from the remote logging system with RELP.

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 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.

Procedure

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

```
---
- name: Configure server-side of the remote logging solution using RELP
  hosts: managed-node-01.example.com
  tasks:
    - name: Deploying remote input and remote_files output
      ansible.builtin.include_role:
        name: redhat.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:
              - '*client.example.com'
        logging_outputs:
          - name: remote_files_output
            type: remote_files
        logging_flows:
          - name: example_flow
            inputs: [relp_server]
            outputs: [remote_files_output]
```

The settings specified in the example playbook include the following:

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 the {**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 the {**ca_cert**, **cert**, **private_key**} triplet is set, files are expected to be on the default path before the logging configuration.

- If both 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 the certificate. 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 certificate file path which is copied to the managed node. If **ca_cert** is specified, it is copied to the location.

cert_src
: Represents the local certificate file path which is copied to the managed node. If **cert** is specified, it is copied to the location.

private_key_src
: Represents the local key file path which is copied to the managed node. 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.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Additional resources

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

- **/usr/share/doc/rhel-system-roles/logging/** directory
- **rsyslog.conf(5)** and **syslog(3)** manual pages

CHAPTER 16. CONFIGURING PERFORMANCE MONITORING WITH PCP BY USING RHEL SYSTEM ROLES

Performance Co-Pilot (PCP) is a system performance analysis toolkit. You can use it to record and analyze performance data from many components on a Red Hat Enterprise Linux system.

You can use the **metrics** RHEL system role to automate the installation and configuration of PCP, and the role can configure Grafana to visualize PCP metrics.

16.1. CONFIGURING PERFORMANCE CO-PILOT BY USING THEMETRICS RHEL SYSTEM ROLE

You can use Performance Co-Pilot (PCP) to monitor many metrics, such as CPU utilization and memory usage. For example, this can help to identify resource and performance bottlenecks. By using the **metrics** RHEL system role, you can remotely configure PCP on multiple hosts to record metrics.

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.

Procedure

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

```
---
- name: Monitoring performance metrics
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure Performance Co-Pilot
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.metrics
      vars:
        metrics_retention_days: 14
        metrics_manage_firewall: true
        metrics_manage_selinux: true
```

The settings specified in the example playbook include the following:

metrics_retention_days: *<number>*

Sets the number of days after which the **pmlogger_daily** systemd timer removes old PCP archives.

metrics_manage_firewall: *<true|false>*

Defines whether the role should open the required ports in the **firewalld** service. If you want to remotely access PCP on the managed nodes, set this variable to **true**.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Query a metric, for example:

```
# ansible managed-node-01.example.com -m command -a 'pminfo -f kernel.all.load'
```

Next step

- Optional: Configure Grafana to monitor PCP hosts and visualize metrics .

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** file
- **/usr/share/doc/rhel-system-roles/metrics/** directory

16.2. CONFIGURING PERFORMANCE CO-PILOT WITH AUTHENTICATION BY USING THE METRICS RHEL SYSTEM ROLE

You can enable authentication in Performance Co-Pilot (PCP) so that the **pmcd** service and Performance Metrics Domain Agents (PDMAs) can determine whether the user running the monitoring tools is allowed to perform an action. Authenticated users have access to metrics with sensitive information. Additionally, certain agents require authentication. For example, the **bpftrace** agent uses authentication to identify whether a user is allowed to load **bpftrace** scripts into the kernel to generate metrics.

By using the **metrics** RHEL system role, you can remotely configure PCP with authentication on multiple hosts.

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.

Procedure

1. Store your sensitive variables in an encrypted file:

   a. Create the vault:

```
$ ansible-vault create ~/vault.yml
New Vault password: <vault_password>
Confirm New Vault password: <vault_password>
```

b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

```
metrics_usr: <username>
metrics_pwd: <password>
```

c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

```
---
- name: Monitoring performance metrics
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Configure Performance Co-Pilot
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.metrics
      vars:
        metrics_retention_days: 14
        metrics_manage_firewall: true
        metrics_manage_selinux: true
        metrics_username: "{{ metrics_usr }}"
        metrics_password: "{{ metrics_pwd }}"
```

The settings specified in the example playbook include the following:

metrics_retention_days: *<number>*

Sets the number of days after which the **pmlogger_daily** systemd timer removes old PCP archives.

metrics_manage_firewall: *<true|false>*

Defines whether the role should open the required ports in the **firewalld** service. If you want to remotely access PCP on the managed nodes, set this variable to **true**.

metrics_username: *<username>*

The role creates this user locally on the managed node, adds the credentials to the **/etc/pcp/passwd.db** Simple Authentication and Security Layer (SASL) database, and configures authentication in PCP. Additionally, if you set **metrics_from_bpftrace: true** in the playbook, PCP uses this account to register **bpftrace** scripts.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --ask-vault-pass --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Verification

- On a host with the **pcp** package installed, query a metric that requires authentication:
  a. Query the metrics by using the credentials that you used in the playbook:

```
# pminfo -fmdt -h pcp://managed-node-01.example.com?username=<user>
proc.fd.count
Password: <password>

proc.fd.count
   inst [844 or "000844 /var/lib/pcp/pmdas/proc/pmdaproc"] value 5
```

  If the command succeeds, it returns the value of the **proc.fd.count** metric.

  b. Run the command again, but omit the username to verify that the command fails for unauthenticated users:

```
# pminfo -fmdt -h pcp://managed-node-01.example.com proc.fd.count

proc.fd.count
Error: No permission to perform requested operation
```

Next step

- Optional: Configure Grafana to monitor PCP hosts and visualize metrics .

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** file
- **/usr/share/doc/rhel-system-roles/metrics/** directory
- Ansible vault

16.3. SETTING UP GRAFANA BY USING THE METRICS RHEL SYSTEM ROLE TO MONITOR MULTIPLE HOSTS WITH PERFORMANCE CO-PILOT

If you have already configured Performance Co-Pilot (PCP) on multiple hosts, you can use an instance of Grafana to visualize the metrics for these hosts. You can display the live data and, if the PCP data is stored in a Redis database, also past data.

By using the **metrics** RHEL system role, you can automate the process of setting up Grafana, the PCP plug-in, the optional Redis database, and the configuration of the data sources.

NOTE

If you use the **metrics** role to install Grafana on a host, the role also installs automatically PCP on this host.

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.
- PCP is configured for remote access on the hosts you want to monitor .
- The host on which you want to install Grafana can access port 44321 on the PCP nodes you plan to monitor.

Procedure

1. Store your sensitive variables in an encrypted file:

   a. Create the vault:

   ```
   $ ansible-vault create ~/vault.yml
   New Vault password: <vault_password>
   Confirm New Vault password: <vault_password>
   ```

   b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

   ```
   grafana_admin_pwd: <password>
   ```

   c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

```
---
- name: Monitoring performance metrics
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Set up Grafana to monitor multiple hosts
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.metrics
      vars:
        metrics_graph_service: true
        metrics_query_service: true
        metrics_monitored_hosts:
          - <pcp_host_1.example.com>
          - <pcp_host_2.example.com>
        metrics_manage_firewall: true
        metrics_manage_selinux: true
```

```
  - name: Set Grafana admin password
    ansible.builtin.shell:
      cmd: grafana-cli admin reset-admin-password "{{ grafana_admin_pwd }}"
```

The settings specified in the example playbook include the following:

metrics_graph_service: true
: Installs Grafana and the PCP plug-in. Additionally, the role adds the **PCP Vector**, **PCP Redis**, and **PCP bpftrace** data sources to Grafana.

metrics_query_service: *<true|false>*
: Defines whether the role should install and configure Redis for centralized metric recording. If enabled, data collected from PCP clients is stored in Redis and, as a result, you can also display historical data instead of only live data.

metrics_monitored_hosts: *<list_of_hosts>*
: Defines the list of hosts to monitor. In Grafana, you can then display the data of these hosts and, additionally, the host that runs Grafana.

metrics_manage_firewall: *<true|false>*
: Defines whether the role should open the required ports in the **firewalld** service. If you set this variable to **true**, you can, for example, access Grafana remotely.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --ask-vault-pass --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Verification

1. Open **http://*<grafana_server_IP_or_hostname>*:3000** in your browser, and log in as the **admin** user with the password you set in the procedure.

2. Display monitoring data:

- To display live data:

   i. Click the **Performance Co-Pilot** icon in the navigation bar on the left, and select **PCP Vector Checklist**.

   ii. By default, the graphs display metrics from the host that runs Grafana. To switch to a different host, enter the hostname in the **hostspec** field and press **Enter**.

- To display historical data stored in a Redis database: Create a panel with a PCP Redis data source. This requires that you set **metrics_query_service: true** in the playbook.

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** file
- **/usr/share/doc/rhel-system-roles/metrics/** directory
- Ansible vault

CHAPTER 17. CONFIGURING NBDE BY USING RHEL SYSTEM ROLES

You can use the **nbde_client** and **nbde_server** RHEL system roles for automated deployments of Policy-Based Decryption (PBD) solutions using Clevis and Tang. The **rhel-system-roles** package contains these system roles, the related examples, and also the reference documentation.

17.1. USING THE NBDE_SERVER RHEL SYSTEM ROLE FOR SETTING UP MULTIPLE TANG SERVERS

By using the **nbde_server** 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

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.

Procedure

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

```
---
- name: Deploy a Tang server
  hosts: tang.server.example.com
  tasks:
  - name: Install and configure periodic key rotation
    ansible.builtin.include_role:
        name: redhat.rhel_system_roles.nbde_server
    vars:
      nbde_server_rotate_keys: yes
      nbde_server_manage_firewall: true
      nbde_server_manage_selinux: true
```

This example playbook ensures deploying of your Tang server and a key rotation.

The settings specified in the example playbook include the following:

nbde_server_manage_firewall: true
: Use the **firewall** system role to manage ports used by the **nbde_server** role.

nbde_server_manage_selinux: true
: Use the **selinux** system role to manage ports used by the **nbde_server** role.
For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.nbde_server/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- On your NBDE client, verify that your Tang server works correctly by using the following command. The command must return the identical message you pass for encryption and decryption:

```
# ansible managed-node-01.example.com -m command -a 'echo test | clevis encrypt tang
'{"url":"<tang.server.example.com>"}' -y | clevis decrypt'
test
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.nbde_server/README.md** file
- **/usr/share/doc/rhel-system-roles/nbde_server/** directory

17.2. SETTING UP CLEVIS CLIENTS WITH DHCP BY USING THE NBDE_CLIENT RHEL SYSTEM ROLE

The **nbde_client** system role enables you to deploy multiple Clevis clients in an automated way.

This role supports binding 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.

Policy-Based Decryption (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.

NOTE

The **nbde_client** system role supports only Tang bindings. Therefore, you cannot use it for TPM2 bindings.

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.
- A volume that is already encrypted by using LUKS.

Procedure

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

```
---
- name: Configure clients for unlocking of encrypted volumes by Tang servers
  hosts: managed-node-01.example.com
  tasks:
    - name: Create NBDE client bindings
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.nbde_client
      vars:
        nbde_client_bindings:
          - device: /dev/rhel/root
            encryption_key_src: /etc/luks/keyfile
            nbde_client_early_boot: true
            state: present
            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
```

   This example playbook configures Clevis clients for automated unlocking of two LUKS-encrypted volumes when at least one of two Tang servers is available.

   The settings specified in the example playbook include the following:

   state: present

   The values of **state** indicate the configuration after you run the playbook. 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.

   nbde_client_early_boot: true

   The **nbde_client** role ensures that networking for a Tang pin is available during early boot by default. If you scenario requires to disable this feature, add the **nbde_client_early_boot: false** variable to your playbook.
   For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.nbde_client/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

1. On your NBDE client, check that the encrypted volume that should be automatically unlocked by your Tang servers contain the corresponding information in its LUKS pins:

```
# ansible managed-node-01.example.com -m command -a 'clevis luks list -d /dev/rhel/root'
1: tang '{"url":"<http://server1.example.com/>"}'
2: tang '{"url":"<http://server2.example.com/>"}'
```

2. If you do not use the **nbde_client_early_boot: false** variable, verify that the bindings are available for the early boot, for example:

```
# ansible managed-node-01.example.com -m command -a 'lsinitrd | grep clevis-luks'
lrwxrwxrwx   1 root     root          48 Jan  4 02:56
etc/systemd/system/cryptsetup.target.wants/clevis-luks-askpass.path ->
/usr/lib/systemd/system/clevis-luks-askpass.path
…
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.nbde_client/README.md** file
- **/usr/share/doc/rhel-system-roles/nbde_client/** directory

17.3. SETTING UP STATIC-IP CLEVIS CLIENTS BY USING THE NBDE_CLIENT RHEL SYSTEM ROLE

The **nbde_client** RHEL system role supports only scenarios with Dynamic Host Configuration Protocol (DHCP). On an NBDE client with static IP configuration, you must pass your network configuration as a kernel boot parameter.

Typically, administrators want to reuse a playbook and not maintain individual playbooks for each host to which Ansible assigns static IP addresses during early boot. In this case, you can use variables in the playbook and provide the settings in an external file. As a result, you need only one playbook and one file with the settings.

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.
- A volume that is already encrypted by using LUKS.

Procedure

1. Create a file with the network settings of your hosts, for example, **static-ip-settings-clients.yml**, and add the values you want to dynamically assign to the hosts:

```
clients:
  managed-node-01.example.com:
    ip_v4: 192.0.2.1
    gateway_v4: 192.0.2.254
    netmask_v4: 255.255.255.0
    interface: enp1s0
  managed-node-02.example.com:
    ip_v4: 192.0.2.2
    gateway_v4: 192.0.2.254
    netmask_v4: 255.255.255.0
    interface: enp1s0
```

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

```
- name: Configure clients for unlocking of encrypted volumes by Tang servers
  hosts: managed-node-01.example.com,managed-node-02.example.com
  vars_files:
    - ~/static-ip-settings-clients.yml
  tasks:
    - name: Create NBDE client bindings
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.network
      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

    - name: Configure a Clevis client with static IP address during early boot
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.bootloader
      vars:
        bootloader_settings:
          - kernel: ALL
            options:
              - name: ip
                value: "{{ clients[inventory_hostname]['ip_v4'] }}::{{ clients[inventory_hostname]
['gateway_v4'] }}:{{ clients[inventory_hostname]['netmask_v4'] }}::{{
clients[inventory_hostname]['interface'] }}:none"
```

This playbook reads certain values dynamically for each host listed in the **~/static-ip-settings-clients.yml** file.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.nbde_client/README.md** file
- **/usr/share/doc/rhel-system-roles/nbde_client/** directory
- Looking forward to Linux network configuration in the initial ramdisk (initrd) (Red Hat Enable Sysadmin)

CHAPTER 18. CONFIGURING NETWORK SETTINGS BY USING RHEL SYSTEM ROLES

By using the **network** RHEL system role, you can automate network-related configuration and management tasks.

18.1. CONFIGURING AN ETHERNET CONNECTION WITH A STATIC IP ADDRESS BY USING THE NETWORK RHEL SYSTEM ROLE WITH AN INTERFACE NAME

To connect a Red Hat Enterprise Linux host to an Ethernet network, create a NetworkManager connection profile for the network device. By using Ansible and the **network** RHEL system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

You can use the **network** RHEL system role to configure an Ethernet connection with static IP addresses, gateways, and DNS settings, and assign them to a specified interface name.

Typically, administrators want to reuse a playbook and not maintain individual playbooks for each host to which Ansible should assign static IP addresses. In this case, you can use variables in the playbook and maintain the settings in the inventory. As a result, you need only one playbook to dynamically assign individual settings to multiple hosts.

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.
- A physical or virtual Ethernet device exists in the server configuration.
- The managed nodes use NetworkManager to configure the network.

Procedure

1. Edit the **~/inventory** file, and append the host-specific settings to the host entries:

```
managed-node-01.example.com interface=enp1s0 ip_v4=192.0.2.1/24
ip_v6=2001:db8:1::1/64 gateway_v4=192.0.2.254 gateway_v6=2001:db8:1::fffe

managed-node-02.example.com interface=enp1s0 ip_v4=192.0.2.2/24
ip_v6=2001:db8:1::2/64 gateway_v4=192.0.2.254 gateway_v6=2001:db8:1::fffe
```

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

```
---
- name: Configure the network
  hosts: managed-node-01.example.com,managed-node-02.example.com
  tasks:
    - name: Ethernet connection profile with static IP address settings
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.network
```

```
    vars:
      network_connections:
        - name: "{{ interface }}"
          interface_name: "{{ interface }}"
          type: ethernet
          autoconnect: yes
          ip:
            address:
              - "{{ ip_v4 }}"
              - "{{ ip_v6 }}"
            gateway4: "{{ gateway_v4 }}"
            gateway6: "{{ gateway_v6 }}"
            dns:
              - 192.0.2.200
              - 2001:db8:1::ffbb
            dns_search:
              - example.com
          state: up
```

This playbook reads certain values dynamically for each host from the inventory file and uses static values in the playbook for settings which are the same for all hosts.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Query the Ansible facts of the managed node and verify the active network settings:

```
# ansible managed-node-01.example.com -m ansible.builtin.setup
...
        "ansible_default_ipv4": {
            "address": "192.0.2.1",
            "alias": "enp1s0",
            "broadcast": "192.0.2.255",
            "gateway": "192.0.2.254",
            "interface": "enp1s0",
            "macaddress": "52:54:00:17:b8:b6",
            "mtu": 1500,
            "netmask": "255.255.255.0",
            "network": "192.0.2.0",
            "prefix": "24",
            "type": "ether"
        },
```

```
        "ansible_default_ipv6": {
            "address": "2001:db8:1::1",
            "gateway": "2001:db8:1::fffe",
            "interface": "enp1s0",
            "macaddress": "52:54:00:17:b8:b6",
            "mtu": 1500,
            "prefix": "64",
            "scope": "global",
            "type": "ether"
        },
        ...
        "ansible_dns": {
            "nameservers": [
                "192.0.2.1",
                "2001:db8:1::ffbb"
            ],
            "search": [
                "example.com"
            ]
        },
...
```

Additional resources

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

18.2. CONFIGURING AN ETHERNET CONNECTION WITH A STATIC IP ADDRESS BY USING THE NETWORK RHEL SYSTEM ROLE WITH A DEVICE PATH

To connect a Red Hat Enterprise Linux host to an Ethernet network, create a NetworkManager connection profile for the network device. By using Ansible and the **network** RHEL system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

You can use the **network** RHEL system role to configure an Ethernet connection with static IP addresses, gateways, and DNS settings, and assign them to a device based on its path instead of its name.

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.
- A physical or virtual Ethernet device exists in the server’s configuration.
- The managed nodes use NetworkManager to configure the network.
- You know the path of the device. You can display the device path by using the **udevadm info /sys/class/net/*<device_name>* | grep ID_PATH=** command.

Procedure

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

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Ethernet connection profile with static IP address settings
      ansible.builtin.include_role:
        name: redhat.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 settings specified in the example playbook include the following:

   match
   : Defines that a condition must be met in order to apply the settings. You can only use this variable with the **path** option.

   path
   : Defines the persistent path of a device. You can set it as a fixed path or an expression. Its value can contain modifiers and wildcards. The example applies the settings to devices that match PCI ID **0000:00:0[1-3].0**, but not **0000:00:02.0**.

   For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Query the Ansible facts of the managed node and verify the active network settings:

```
# ansible managed-node-01.example.com -m ansible.builtin.setup
...
        "ansible_default_ipv4": {
            "address": "192.0.2.1",
            "alias": "enp1s0",
            "broadcast": "192.0.2.255",
            "gateway": "192.0.2.254",
            "interface": "enp1s0",
            "macaddress": "52:54:00:17:b8:b6",
            "mtu": 1500,
            "netmask": "255.255.255.0",
            "network": "192.0.2.0",
            "prefix": "24",
            "type": "ether"
        },
        "ansible_default_ipv6": {
            "address": "2001:db8:1::1",
            "gateway": "2001:db8:1::fffe",
            "interface": "enp1s0",
            "macaddress": "52:54:00:17:b8:b6",
            "mtu": 1500,
            "prefix": "64",
            "scope": "global",
            "type": "ether"
        },
        ...
        "ansible_dns": {
            "nameservers": [
                "192.0.2.1",
                "2001:db8:1::ffbb"
            ],
            "search": [
                "example.com"
            ]
        },
...
```

Additional resources

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

18.3. CONFIGURING AN ETHERNET CONNECTION WITH A DYNAMIC IP ADDRESS BY USING THE NETWORK RHEL SYSTEM ROLE WITH AN INTERFACE NAME

To connect a Red Hat Enterprise Linux host to an Ethernet network, create a NetworkManager connection profile for the network device. By using Ansible and the **network** RHEL system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

You can use the **network** RHEL system role to configure an Ethernet connection that retrieves its IP addresses, gateways, and DNS settings from a DHCP server and IPv6 stateless address autoconfiguration (SLAAC). With this role you can assign the connection profile to the specified interface name.

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.
- A physical or virtual Ethernet device exists in the servers' configuration.
- A DHCP server and SLAAC are available in the network.
- The managed nodes use the NetworkManager service to configure the network.

Procedure

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

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Ethernet connection profile with dynamic IP address settings
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.network
      vars:
        network_connections:
          - name: enp1s0
            interface_name: enp1s0
            type: ethernet
            autoconnect: yes
            ip:
              dhcp4: yes
              auto6: yes
            state: up
```

The settings specified in the example playbook include the following:

dhcp4: yes
: Enables automatic IPv4 address assignment from DHCP, PPP, or similar services.

auto6: yes
: Enables IPv6 auto-configuration. By default, NetworkManager uses Router Advertisements. If the router announces the **managed** flag, NetworkManager requests an IPv6 address and prefix from a DHCPv6 server.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Query the Ansible facts of the managed node and verify that the interface received IP addresses and DNS settings:

```
# ansible managed-node-01.example.com -m ansible.builtin.setup
...
        "ansible_default_ipv4": {
            "address": "192.0.2.1",
            "alias": "enp1s0",
            "broadcast": "192.0.2.255",
            "gateway": "192.0.2.254",
            "interface": "enp1s0",
            "macaddress": "52:54:00:17:b8:b6",
            "mtu": 1500,
            "netmask": "255.255.255.0",
            "network": "192.0.2.0",
            "prefix": "24",
            "type": "ether"
        },
        "ansible_default_ipv6": {
            "address": "2001:db8:1::1",
            "gateway": "2001:db8:1::fffe",
            "interface": "enp1s0",
            "macaddress": "52:54:00:17:b8:b6",
            "mtu": 1500,
            "prefix": "64",
            "scope": "global",
            "type": "ether"
        },
        ...
        "ansible_dns": {
            "nameservers": [
                "192.0.2.1",
                "2001:db8:1::ffbb"
            ],
            "search": [
                "example.com"
            ]
        },
...
```

Additional resources

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

18.4. CONFIGURING AN ETHERNET CONNECTION WITH A DYNAMIC IP ADDRESS BY USING THE NETWORK RHEL SYSTEM ROLE WITH A DEVICE PATH

To connect a Red Hat Enterprise Linux host to an Ethernet network, create a NetworkManager connection profile for the network device. By using Ansible and the **network** RHEL system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

You can use the **network** RHEL system role to configure an Ethernet connection that retrieves its IP addresses, gateways, and DNS settings from a DHCP server and IPv6 stateless address autoconfiguration (SLAAC). The role can assign the connection profile to a device based on its path instead of an interface name.

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.
- A physical or virtual Ethernet device exists in the server's configuration.
- A DHCP server and SLAAC are available in the network.
- The managed hosts use NetworkManager to configure the network.
- You know the path of the device. You can display the device path by using the **udevadm info /sys/class/net/<*device_name*> | grep ID_PATH=** command.

Procedure

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

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Ethernet connection profile with dynamic IP address settings
      ansible.builtin.include_role:
        name: redhat.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 settings specified in the example playbook include the following:

match: path
: Defines that a condition must be met in order to apply the settings. You can only use this variable with the **path** option.

path: *<path_and_expressions>*
: Defines the persistent path of a device. You can set it as a fixed path or an expression. Its value can contain modifiers and wildcards. The example applies the settings to devices that match PCI ID **0000:00:0[1-3].0**, but not **0000:00:02.0**.

dhcp4: yes
: Enables automatic IPv4 address assignment from DHCP, PPP, or similar services.

auto6: yes
: Enables IPv6 auto-configuration. By default, NetworkManager uses Router Advertisements. If the router announces the **managed** flag, NetworkManager requests an IPv6 address and prefix from a DHCPv6 server.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Query the Ansible facts of the managed node and verify that the interface received IP addresses and DNS settings:

```
# ansible managed-node-01.example.com -m ansible.builtin.setup
...
        "ansible_default_ipv4": {
            "address": "192.0.2.1",
            "alias": "enp1s0",
            "broadcast": "192.0.2.255",
            "gateway": "192.0.2.254",
            "interface": "enp1s0",
            "macaddress": "52:54:00:17:b8:b6",
            "mtu": 1500,
            "netmask": "255.255.255.0",
            "network": "192.0.2.0",
```

```
            "prefix": "24",
            "type": "ether"
        },
        "ansible_default_ipv6": {
            "address": "2001:db8:1::1",
            "gateway": "2001:db8:1::fffe",
            "interface": "enp1s0",
            "macaddress": "52:54:00:17:b8:b6",
            "mtu": 1500,
            "prefix": "64",
            "scope": "global",
            "type": "ether"
        },
        ...
        "ansible_dns": {
            "nameservers": [
                "192.0.2.1",
                "2001:db8:1::ffbb"
            ],
            "search": [
                "example.com"
            ]
        },
...
```

Additional resources

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

18.5. CONFIGURING A STATIC ETHERNET CONNECTION WITH 802.1X NETWORK AUTHENTICATION BY USING THE NETWORK RHEL SYSTEM ROLE

Network Access Control (NAC) protects a network from unauthorized clients. You can specify the details that are required for the authentication in NetworkManager connection profiles to enable clients to access the network. By using Ansible and the **network** RHEL system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

You can use an Ansible playbook to copy a private key, a certificate, and the CA certificate to the client, and then use the **network** RHEL system role to configure a connection profile with 802.1X network authentication.

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 network supports 802.1X network authentication.

- The managed nodes use NetworkManager.
- The following files required for the 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. Store your sensitive variables in an encrypted file:
   a. Create the vault:

   ```
   $ ansible-vault create ~/vault.yml
   New Vault password: <vault_password>
   Confirm New Vault password: <vault_password>
   ```

   b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

   ```
   pwd: <password>
   ```

   c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

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

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

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

    - name: Ethernet connection profile with static IP address settings and 802.1X
      ansible.builtin.include_role:
        name: redhat.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: "{{ pwd }}"
            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
```

The settings specified in the example playbook include the following:

ieee802_1x
: This variable contains the 802.1X-related settings.

eap: tls
: Configures the profile to use the certificate-based **TLS** authentication method for the Extensible Authentication Protocol (EAP).

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --ask-vault-pass --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Verification

- Access resources on the network that require network authentication.

Additional resources

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

- **/usr/share/doc/rhel-system-roles/network/** directory
- Ansible vault

18.6. CONFIGURING A NETWORK BOND BY USING THE NETWORK RHEL SYSTEM ROLE

You can combine network interfaces in a bond to provide a logical interface with higher throughput or redundancy. To configure a bond, create a NetworkManager connection profile. By using Ansible and the **network** RHEL system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

You can use the **network** RHEL system role to configure a network bond and, if a connection profile for the bond's parent device does not exist, the role can create it as well.

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.
- Two or more physical or virtual network devices are installed on the server.

Procedure

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

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Bond connection profile with two Ethernet ports
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.network
      vars:
        network_connections:
          # Bond profile
          - name: bond0
            type: bond
            interface_name: bond0
            ip:
              dhcp4: yes
              auto6: yes
            bond:
              mode: active-backup
            state: up

          # Port profile for the 1st Ethernet device
          - name: bond0-port1
            interface_name: enp7s0
            type: ethernet
            controller: bond0
            state: up
```

```
    # Port profile for the 2nd Ethernet device
    - name: bond0-port2
      interface_name: enp8s0
      type: ethernet
      controller: bond0
      state: up
```

The settings specified in the example playbook include the following:

type: *<profile_type>*
: Sets the type of the profile to create. The example playbook creates three connection profiles: One for the bond and two for the Ethernet devices.

dhcp4: yes
: Enables automatic IPv4 address assignment from DHCP, PPP, or similar services.

auto6: yes
: Enables IPv6 auto-configuration. By default, NetworkManager uses Router Advertisements. If the router announces the **managed** flag, NetworkManager requests an IPv6 address and prefix from a DHCPv6 server.

mode: *<bond_mode>*
: Sets the bonding mode. Possible values are:

- **balance-rr** (default)
- **active-backup**
- **balance-xor**
- **broadcast**
- **802.3ad**
- **balance-tlb**
- **balance-alb**.

Depending on the mode you set, you need to set additional variables in the playbook.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Temporarily remove the network cable from one of the network devices and check if the other device in the bond is handling the traffic.
  Note that there is no method to properly test link failure events using software utilities. Tools that deactivate connections, such as **nmcli**, show only the bonding driver's ability to handle port configuration changes and not actual link failure events.

Additional resources

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

18.7. CONFIGURING VLAN TAGGING BY USING THE NETWORK RHEL SYSTEM ROLE

If your network uses Virtual Local Area Networks (VLANs) to separate network traffic into logical networks, create a NetworkManager connection profile to configure VLAN tagging. By using Ansible and the **network** RHEL system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

You can use the **network** RHEL system role to configure VLAN tagging and, if a connection profile for the VLAN's parent device does not exist, the role can create it as well.

> **NOTE**
>
> If the VLAN device requires an IP address, default gateway, and DNS settings, configure them on the VLAN device and not on the parent device.

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.

Procedure

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

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: VLAN connection profile with Ethernet port
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.network
      vars:
        network_connections:
          # Ethernet profile
          - name: enp1s0
            type: ethernet
            interface_name: enp1s0
```

```
        autoconnect: yes
        state: up
        ip:
          dhcp4: no
          auto6: no

      # VLAN profile
      - name: enp1s0.10
        type: vlan
        vlan:
          id: 10
        ip:
          dhcp4: yes
          auto6: yes
        parent: enp1s0
        state: up
```

The settings specified in the example playbook include the following:

type: *<profile_type>*

Sets the type of the profile to create. The example playbook creates two connection profiles: One for the parent Ethernet device and one for the VLAN device.

dhcp4: *<value>*

If set to **yes**, automatic IPv4 address assignment from DHCP, PPP, or similar services is enabled. Disable the IP address configuration on the parent device.

auto6: *<value>*

If set to **yes**, IPv6 auto-configuration is enabled. In this case, by default, NetworkManager uses Router Advertisements and, if the router announces the **managed** flag, NetworkManager requests an IPv6 address and prefix from a DHCPv6 server. Disable the IP address configuration on the parent device.

parent: *<parent_device>*

Sets the parent device of the VLAN connection profile. In the example, the parent is the Ethernet interface.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Verify the VLAN settings:

```
# ansible managed-node-01.example.com -m command -a 'ip -d addr show enp1s0.10'
```

```
managed-node-01.example.com | CHANGED | rc=0 >>
4: vlan10@enp1s0.10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
noqueue state UP group default qlen 1000
    link/ether 52:54:00:72:2f:6e brd ff:ff:ff:ff:ff:ff promiscuity 0
    vlan protocol 802.1Q id 10 <REORDER_HDR> numtxqueues 1 numrxqueues 1
gso_max_size 65536 gso_max_segs 65535
    ...
```

Additional resources

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

18.8. CONFIGURING A NETWORK BRIDGE BY USING THE NETWORK RHEL SYSTEM ROLE

You can connect multiple networks on layer 2 of the Open Systems Interconnection (OSI) model by creating a network bridge. To configure a bridge, create a connection profile in NetworkManager. By using Ansible and the **network** RHEL system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

You can use the **network** RHEL system role to configure a bridge and, if a connection profile for the bridge's parent device does not exist, the role can create it as well.

> **NOTE**
>
> If you want to assign IP addresses, gateways, and DNS settings to a bridge, configure them on the bridge and not on its ports.

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.
- Two or more physical or virtual network devices are installed on the server.

Procedure

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

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Bridge connection profile with two Ethernet ports
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.network
      vars:
        network_connections:
          # Bridge profile
```

```
        - name: bridge0
          type: bridge
          interface_name: bridge0
          ip:
            dhcp4: yes
            auto6: yes
          state: up

        # Port profile for the 1st Ethernet device
        - name: bridge0-port1
          interface_name: enp7s0
          type: ethernet
          controller: bridge0
          port_type: bridge
          state: up

        # Port profile for the 2nd Ethernet device
        - name: bridge0-port2
          interface_name: enp8s0
          type: ethernet
          controller: bridge0
          port_type: bridge
          state: up
```

The settings specified in the example playbook include the following:

type: *<profile_type>*
: Sets the type of the profile to create. The example playbook creates three connection profiles: One for the bridge and two for the Ethernet devices.

dhcp4: yes
: Enables automatic IPv4 address assignment from DHCP, PPP, or similar services.

auto6: yes
: Enables IPv6 auto-configuration. By default, NetworkManager uses Router Advertisements. If the router announces the **managed** flag, NetworkManager requests an IPv6 address and prefix from a DHCPv6 server.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

1. Display the link status of Ethernet devices that are ports of a specific bridge:

```
# ansible managed-node-01.example.com -m command -a 'ip link show master bridge0'
managed-node-01.example.com | CHANGED | rc=0 >>
3: enp7s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master bridge0 state UP mode DEFAULT group default qlen 1000
    link/ether 52:54:00:62:61:0e brd ff:ff:ff:ff:ff:ff
4: enp8s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master bridge0 state UP mode DEFAULT group default qlen 1000
    link/ether 52:54:00:9e:f1:ce brd ff:ff:ff:ff:ff:ff
```

2. Display the status of Ethernet devices that are ports of any bridge device:

```
# ansible managed-node-01.example.com -m command -a 'bridge link show'
managed-node-01.example.com | CHANGED | rc=0 >>
3: enp7s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master bridge0 state forwarding priority 32 cost 100
4: enp8s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master bridge0 state listening priority 32 cost 100
```

Additional resources

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

18.9. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION BY USING THE NETWORK RHEL SYSTEM ROLE

A host forwards a network packet to its default gateway if the packet's destination can neither be reached through the directly-connected networks nor through any of the routes configured on the host. To configure the default gateway of a host, set it in the NetworkManager connection profile of the interface that is connected to the same network as the default gateway. By using Ansible and the **network** RHEL system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

In most situations, administrators set the default gateway when they create a connection. However, you can also set or update the default gateway setting on a previously-created connection.

> **WARNING**
>
> You cannot use the **network** RHEL system role to update only specific values in an existing connection profile. The role ensures that a connection profile exactly matches the settings in a playbook. If a connection profile with the same name already exists, the role applies the settings from the playbook and resets all other settings in the profile to their defaults. To prevent resetting values, always specify the whole configuration of the network connection profile in the playbook, including the settings that you do not want to change.

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.

Procedure

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

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Ethernet connection profile with static IP address settings
      ansible.builtin.include_role:
        name: redhat.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
```

   For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Query the Ansible facts of the managed node and verify the active network settings:

```
# ansible managed-node-01.example.com -m ansible.builtin.setup
...
```

```
    "ansible_default_ipv4": {
  ...
       "gateway": "198.51.100.254",
       "interface": "enp1s0",
  ...
    },
    "ansible_default_ipv6": {
  ...
       "gateway": "2001:db8:1::fffe",
       "interface": "enp1s0",
  ...
 }
...
```

Additional resources

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

18.10. CONFIGURING A STATIC ROUTE BY USING THENETWORK RHEL SYSTEM ROLE

A static route ensures that you can send traffic to a destination that cannot be reached through the default gateway. You configure static routes in the NetworkManager connection profile of the interface that is connected to the same network as the next hop. By using Ansible and the **network** RHEL system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

> **WARNING**
>
> You cannot use the **network** RHEL system role to update only specific values in an existing connection profile. The role ensures that a connection profile exactly matches the settings in a playbook. If a connection profile with the same name already exists, the role applies the settings from the playbook and resets all other settings in the profile to their defaults. To prevent resetting values, always specify the whole configuration of the network connection profile in the playbook, including the settings that you do not want to change.

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.

Procedure

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

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Ethernet connection profile with static IP address settings
      ansible.builtin.include_role:
        name: redhat.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
```

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

1. Display the IPv4 routes:

```
# ansible managed-node-01.example.com -m command -a 'ip -4 route'
managed-node-01.example.com | CHANGED | rc=0 >>
...
```

```
198.51.100.0/24 via 192.0.2.10 dev enp7s0
```

2. Display the IPv6 routes:

```
# ansible managed-node-01.example.com -m command -a 'ip -6 route'
managed-node-01.example.com | CHANGED | rc=0 >>
...
2001:db8:2::/64 via 2001:db8:1::10 dev enp7s0 metric 1024 pref medium
```

Additional resources

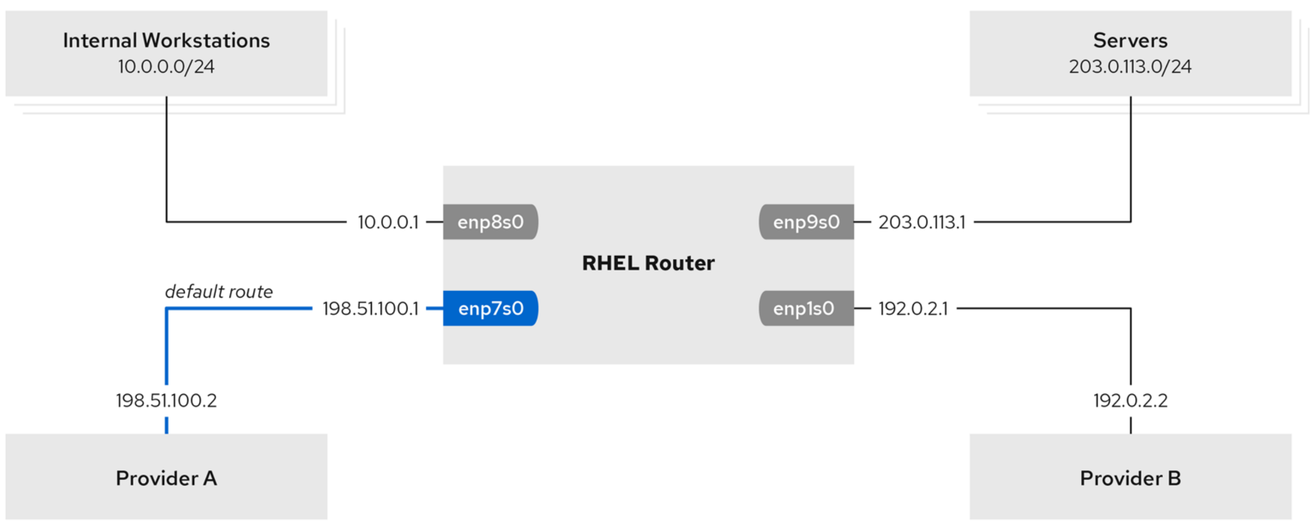
- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file
- **/usr/share/doc/rhel-system-roles/network/** directory

18.11. ROUTING TRAFFIC FROM A SPECIFIC SUBNET TO A DIFFERENT DEFAULT GATEWAY BY USING THE NETWORK RHEL SYSTEM ROLE

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. By using Ansible and the **network** RHEL system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

You can use the **network** RHEL system role to configure the connection profiles, including routing tables and rules.

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 them.

- The managed nodes use NetworkManager and the **firewalld** service.
- 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 ~/**playbook.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
      ansible.builtin.include_role:
        name: redhat.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
```

The settings specified in the example playbook include the following:

table: *<value>*
: Assigns the route from the same list entry as the **table** variable to the specified routing table.

routing_rule: *<list>*
: Defines the priority of the specified routing rule and from a connection profile to which routing table the rule is assigned.

zone: *<zone_name>*
: Assigns the network interface from a connection profile to the specified **firewalld** zone.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.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  192.0.2.2 (192.0.2.2)   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** file
- **/usr/share/doc/rhel-system-roles/network/** directory

18.12. CONFIGURING AN ETHTOOL OFFLOAD FEATURE BY USING THE NETWORK RHEL SYSTEM ROLE

Network interface controllers can use the TCP offload engine (TOE) to offload processing certain operations to the network controller. This improves the network throughput. You configure offload features in the connection profile of the network interface. By using Ansible and the **network** RHEL system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

> **WARNING**
>
> You cannot use the **network** RHEL system role to update only specific values in an existing connection profile. The role ensures that a connection profile exactly matches the settings in a playbook. If a connection profile with the same name already exists, the role applies the settings from the playbook and resets all other settings in the profile to their defaults. To prevent resetting values, always specify the whole configuration of the network connection profile in the playbook, including the settings that you do not want to change.

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.

Procedure

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

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Ethernet connection profile with dynamic IP address settings and offload features
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.network
      vars:
        network_connections:
          - name: enp1s0
            type: ethernet
            autoconnect: yes
            ip:
              dhcp4: yes
              auto6: yes
            ethtool:
              features:
                gro: no
                gso: yes
                tx_sctp_segmentation: no
            state: up
```

The settings specified in the example playbook include the following:

gro: no
: Disables Generic receive offload (GRO).

gso: yes
: Enables Generic segmentation offload (GSO).

tx_sctp_segmentation: no
Disables TX stream control transmission protocol (SCTP) segmentation.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Query the Ansible facts of the managed node and verify the offload settings:

```
# ansible managed-node-01.example.com -m ansible.builtin.setup
...
        "ansible_enp1s0": {
            "active": true,
            "device": "enp1s0",
     "features": {
          ...
  "rx_gro_hw": "off,
          ...
  "tx_gso_list": "on,
          ...
  "tx_sctp_segmentation": "off",
  ...
            }
...
```

Additional resources

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

18.13. CONFIGURING AN ETHTOOL COALESCE SETTINGS BY USING THE NETWORK RHEL SYSTEM ROLE

By using interrupt coalescing, the system collects network packets and generates a single interrupt for multiple packets. This increases the amount of data sent to the kernel with one hardware interrupt, which reduces the interrupt load, and maximizes the throughput. You configure coalesce settings in the connection profile of the network interface. By using Ansible and the **network** RHEL role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

> **WARNING**
>
> You cannot use the **network** RHEL system role to update only specific values in an existing connection profile. The role ensures that a connection profile exactly matches the settings in a playbook. If a connection profile with the same name already exists, the role applies the settings from the playbook and resets all other settings in the profile to their defaults. To prevent resetting values, always specify the whole configuration of the network connection profile in the playbook, including the settings that you do not want to change.

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.

Procedure

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

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Ethernet connection profile with dynamic IP address settings and coalesce
settings
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.network
      vars:
        network_connections:
          - name: enp1s0
            type: ethernet
            autoconnect: yes
            ip:
              dhcp4: yes
              auto6: yes
            ethtool:
              coalesce:
                rx_frames: 128
                tx_frames: 128
            state: up
```

The settings specified in the example playbook include the following:

rx_frames: *<value>*
: Sets the number of RX frames.

gso: *<value>*
: Sets the number of TX frames.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Display the current offload features of the network device:

```
# ansible managed-node-01.example.com -m command -a 'ethtool -c enp1s0'
managed-node-01.example.com | CHANGED | rc=0 >>
...
rx-frames: 128
...
tx-frames: 128
...
```

Additional resources

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

18.14. INCREASING THE RING BUFFER SIZE TO REDUCE A HIGH PACKET DROP RATE BY USING THE NETWORK RHEL SYSTEM ROLE

Increase the size of an Ethernet device's ring buffers if the packet drop rate causes applications to report a loss of data, timeouts, or other issues.

Ring buffers are circular buffers where an overflow overwrites existing data. The network card assigns a transmit (TX) and receive (RX) ring buffer. Receive ring buffers are shared between the device driver and the network interface controller (NIC). Data can move from NIC to the kernel through either hardware interrupts or software interrupts, also called SoftIRQs.

The kernel uses the RX ring buffer to store incoming packets until the device driver can process them. The device driver drains the RX ring, typically by using SoftIRQs, which puts the incoming packets into a kernel data structure called an **sk_buff** or **skb** to begin its journey through the kernel and up to the application that owns the relevant socket.

The kernel uses the TX ring buffer to hold outgoing packets which should be sent to the network. These ring buffers reside at the bottom of the stack and are a crucial point at which packet drop can occur, which in turn will adversely affect network performance.

You configure ring buffer settings in the NetworkManager connection profiles. By using Ansible and the **network** RHEL system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

> **WARNING**
>
> You cannot use the **network** RHEL system role to update only specific values in an existing connection profile. The role ensures that a connection profile exactly matches the settings in a playbook. If a connection profile with the same name already exists, the role applies the settings from the playbook and resets all other settings in the profile to their defaults. To prevent resetting values, always specify the whole configuration of the network connection profile in the playbook, including the settings that you do not want to change.

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.
- You know the maximum ring buffer sizes that the device supports.

Procedure

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

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Ethernet connection profile with dynamic IP address setting and increased ring
buffer sizes
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.network
      vars:
        network_connections:
          - name: enp1s0
            type: ethernet
            autoconnect: yes
            ip:
              dhcp4: yes
              auto6: yes
            ethtool:
              ring:
                rx: 4096
                tx: 4096
            state: up
```

The settings specified in the example playbook include the following:

rx: *<value>*
: Sets the maximum number of received ring buffer entries.

tx: *<value>*
: Sets the maximum number of transmitted ring buffer entries.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Display the maximum ring buffer sizes:

```
# ansible managed-node-01.example.com -m command -a 'ethtool -g enp1s0'
managed-node-01.example.com | CHANGED | rc=0 >>
...
Current hardware settings:
RX:             4096
RX Mini:        0
RX Jumbo:       0
TX:             4096
```

Additional resources

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

18.15. CONFIGURING AN IPOIB CONNECTION BY USING THE NETWORK RHEL SYSTEM ROLE

You can use IP over InfiniBand (IPoIB) to send IP packets over an InfiniBand interface. To configure IPoIB, create a NetworkManager connection profile. By using Ansible and the **network** system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

You can use the **network** RHEL system role to configure IPoIB and, if a connection profile for the InfiniBand's parent device does not exist, the role can create it as well.

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.
- 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 **~/playbook.yml**, with the following content:

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: IPoIB connection profile with static IP address settings
      ansible.builtin.include_role:
        name: redhat.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
```

The settings specified in the example playbook include the following:

type: *<profile_type>*

Sets the type of the profile to create. The example playbook creates two connection profiles: One for the InfiniBand connection and one for the IPoIB device.

parent: *<parent_device>*

Sets the parent device of the IPoIB connection profile.

p_key: *<value>*

Sets the InfiniBand partition key. If you set this variable, do not set **interface_name** on the IPoIB device.

transport_mode: *<mode>*

Sets the IPoIB connection operation mode. You can set this variable to **datagram** (default) or **connected**.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

1. Display the IP settings of the **mlx4_ib0.8002** device:

```
# ansible managed-node-01.example.com -m command -a 'ip address show
mlx4_ib0.8002'
managed-node-01.example.com | CHANGED | rc=0 >>
...
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:

```
# ansible managed-node-01.example.com -m command -a 'cat
/sys/class/net/mlx4_ib0.8002/pkey'
managed-node-01.example.com | CHANGED | rc=0 >>
0x8002
```

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

```
# ansible managed-node-01.example.com -m command -a 'cat
/sys/class/net/mlx4_ib0.8002/mode'
managed-node-01.example.com | CHANGED | rc=0 >>
datagram
```

Additional resources

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

18.16. 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.

IMPORTANT

You can set only Nmstate YAML instructions in **network_state**. These instructions differ from the variables you can set in **network_connections**.

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

<table>
<tr><td>Playbook with state configurations</td><td>Regular playbook</td></tr>
<tr><td><pre>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</pre></td><td><pre>vars:
  network_connections:
    - name: enp7s0
      interface_name: enp7s0
      type: ethernet
      autoconnect: yes
      ip:
        dhcp4: yes
        auto6: yes
      state: up</pre></td></tr>
</table>

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	Regular playbook

<pre>vars:
  network_state:
    interfaces:
    - name: enp7s0
      type: ethernet
      state: down</pre>
	<pre>vars:
	  network_connections:
	    - name: enp7s0
	      interface_name: enp7s0
	      type: ethernet
	      autoconnect: yes
	      ip:
	        dhcp4: yes
	        auto6: yes
	      state: down</pre>

Additional resources

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

CHAPTER 19. MANAGING CONTAINERS BY USING RHEL SYSTEM ROLES

With the **podman** RHEL system role, you can manage Podman configuration, containers, and **systemd** services that run Podman containers.

19.1. CREATING A ROOTLESS CONTAINER WITH BIND MOUNT BY USING THE PODMAN RHEL SYSTEM ROLE

You can use the **podman** RHEL system role to create rootless containers with bind mount by running an Ansible playbook and with that, manage your application configuration.

The example Ansible playbook starts two Kubernetes pods: one for a database and another for a web application. The database pod configuration is specified in the playbook, while the web application pod is defined in an external YAML file.

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 user and group **webapp** exist, and must be listed in the **/etc/subuid** and **/etc/subgid** files on the host.

Procedure

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

```
- name: Configure Podman
  hosts: managed-node-01.example.com
  tasks:
    - name: Create a web application and a database
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.podman
      vars:
        podman_create_host_directories: true
        podman_firewall:
          - port: 8080-8081/tcp
            state: enabled
          - port: 12340/tcp
            state: enabled
        podman_selinux_ports:
          - ports: 8080-8081
            setype: http_port_t
        podman_kube_specs:
          - state: started
            run_as_user: dbuser
            run_as_group: dbgroup
            kube_file_content:
              apiVersion: v1
              kind: Pod
```

```
          metadata:
            name: db
          spec:
            containers:
              - name: db
                image:  quay.io/linux-system-roles/mysql:5.6
                ports:
                  - containerPort: 1234
                    hostPort: 12340
                volumeMounts:
                  - mountPath: /var/lib/db:Z
                    name: db
            volumes:
              - name: db
                hostPath:
                  path: /var/lib/db
      - state: started
        run_as_user: webapp
        run_as_group: webapp
        kube_file_src: /path/to/webapp.yml
```

The settings specified in the example playbook include the following:

run_as_user and **run_as_group**
: Specify that containers are rootless.

kube_file_content
: Contains a Kubernetes YAML file defining the first container named **db**. You can generate the Kubernetes YAML file by using the **podman kube generate** command.

  - The **db** container is based on the **quay.io/db/db:stable** container image.
  - The **db** bind mount maps the **/var/lib/db** directory on the host to the **/var/lib/db** directory in the container. The **Z** flag labels the content with a private unshared label, therefore, only the **db** container can access the content.

kube_file_src: *<path>*
: Defines the second container. The content of the **/path/to/webapp.yml** file on the controller node will be copied to the **kube_file** field on the managed node.

volumes: *<list>*
: A YAML list to define the source of the data to provide in one or more containers. For example, a local disk on the host (**hostPath**) or other disk device.

volumeMounts: *<list>*
: A YAML list to define the destination where the individual container will mount a given volume.

podman_create_host_directories: true
: Creates the directory on the host. This instructs the role to check the kube specification for **hostPath** volumes and create those directories on the host. If you need more control over the ownership and permissions, use **podman_host_directories**.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.podman/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check --ask-vault-pass ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.podman/README.md** file
- **/usr/share/doc/rhel-system-roles/podman/** directory

19.2. CREATING A ROOTFUL CONTAINER WITH PODMAN VOLUME BY USING THE PODMAN RHEL SYSTEM ROLE

You can use the **podman** RHEL system role to create a rootful container with a Podman volume by running an Ansible playbook and with that, manage your application configuration.

The example Ansible playbook deploys a Kubernetes pod named **ubi8-httpd** running an HTTP server container from the **registry.access.redhat.com/ubi8/httpd-24** image. The container's web content is mounted from a persistent volume named **ubi8-html-volume**. By default, the **podman** role creates rootful containers.

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.

Procedure

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

```
- name: Configure Podman
  hosts: managed-node-01.example.com
  tasks:
    - name: Start Apache server on port 8080
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.podman
  vars:
    podman_firewall:
      - port: 8080/tcp
        state: enabled
    podman_kube_specs:
      - state: started
        kube_file_content:
          apiVersion: v1
          kind: Pod
          metadata:
```

```
        name: ubi8-httpd
      spec:
        containers:
          - name: ubi8-httpd
            image: registry.access.redhat.com/ubi8/httpd-24
            ports:
              - containerPort: 8080
                hostPort: 8080
            volumeMounts:
              - mountPath: /var/www/html:Z
                name: ubi8-html
        volumes:
          - name: ubi8-html
            persistentVolumeClaim:
              claimName: ubi8-html-volume
```

The settings specified in the example playbook include the following:

kube_file_content

Contains a Kubernetes YAML file defining the first container named **db**. You can generate the Kubernetes YAML file by using the **podman kube generate** command.

- The **ubi8-httpd** container is based on the **registry.access.redhat.com/ubi8/httpd-24** container image.
- The **ubi8-html-volume** maps the **/var/www/html** directory on the host to the container. The **Z** flag labels the content with a private unshared label, therefore, only the **ubi8-httpd** container can access the content.
- The pod mounts the existing persistent volume named **ubi8-html-volume** with the mount path **/var/www/html**.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.podman/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.podman/README.md** file
- **/usr/share/doc/rhel-system-roles/podman/** directory

19.3. CREATING A QUADLET APPLICATION WITH SECRETS BY USING THE PODMAN RHEL SYSTEM ROLE

You can use the **podman** RHEL system role to create a Quadlet application with secrets by running an Ansible playbook.

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 certificate and the corresponding private key that the web server in the container should use are stored in the **~/certificate.pem** and **~/key.pem** files.

Procedure

1. Display the contents of the certificate and private key files:

```
$ cat ~/certificate.pem
-----BEGIN CERTIFICATE-----
...
-----END CERTIFICATE-----

$ cat ~/key.pem
-----BEGIN PRIVATE KEY-----
...
-----END PRIVATE KEY-----
```

   You require this information in a later step.

2. Store your sensitive variables in an encrypted file:

   a. Create the vault:

```
$ ansible-vault create ~/vault.yml
New Vault password: <vault_password>
Confirm New Vault password: <vault_password>
```

   b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

```
root_password: <root_password>
certificate: |-
  -----BEGIN CERTIFICATE-----
  ...
  -----END CERTIFICATE-----
key: |-
  -----BEGIN PRIVATE KEY-----
  ...
  -----END PRIVATE KEY-----
```

   Ensure that all lines in the **certificate** and **key** variables start with two spaces.

c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

```
- name: Deploy a wordpress CMS with MySQL database
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
  - name: Create and run the container
    ansible.builtin.include_role:
      name: redhat.rhel_system_roles.podman
    vars:
      podman_create_host_directories: true
      podman_activate_systemd_unit: false
      podman_quadlet_specs:
        - name: quadlet-demo
          type: network
          file_content: |
            [Network]
            Subnet=192.168.30.0/24
            Gateway=192.168.30.1
            Label=app=wordpress
        - file_src: quadlet-demo-mysql.volume
        - template_src: quadlet-demo-mysql.container.j2
        - file_src: envoy-proxy-configmap.yml
        - file_src: quadlet-demo.yml
        - file_src: quadlet-demo.kube
          activate_systemd_unit: true
      podman_firewall:
        - port: 8000/tcp
          state: enabled
        - port: 9000/tcp
          state: enabled
      podman_secrets:
        - name: mysql-root-password-container
          state: present
          skip_existing: true
          data: "{{ root_password }}"
        - name: mysql-root-password-kube
          state: present
          skip_existing: true
          data: |
            apiVersion: v1
            data:
              password: "{{ root_password | b64encode }}"
            kind: Secret
            metadata:
              name: mysql-root-password-kube
        - name: envoy-certificates
          state: present
          skip_existing: true
          data: |
            apiVersion: v1
            data:
              certificate.key: {{ key | b64encode }}
```

```
      certificate.pem: {{ certificate | b64encode }}
    kind: Secret
    metadata:
      name: envoy-certificates
```

The procedure creates a WordPress content management system paired with a MySQL database. The **podman_quadlet_specs role** variable defines a set of configurations for the Quadlet, which refers to a group of containers or services that work together in a certain way. It includes the following specifications:

- The Wordpress network is defined by the **quadlet-demo** network unit.
- The volume configuration for MySQL container is defined by the **file_src: quadlet-demo-mysql.volume** field.
- The **template_src: quadlet-demo-mysql.container.j2** field is used to generate a configuration for the MySQL container.
- Two YAML files follow: **file_src: envoy-proxy-configmap.yml** and **file_src: quadlet-demo.yml**. Note that .yml is not a valid Quadlet unit type, therefore these files will just be copied and not processed as a Quadlet specification.
- The Wordpress and envoy proxy containers and configuration are defined by the **file_src: quadlet-demo.kube** field. The kube unit refers to the previous YAML files in the **[Kube]** section as **Yaml=quadlet-demo.yml** and **ConfigMap=envoy-proxy-configmap.yml**.

4. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check --ask-vault-pass ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

5. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.podman/README.md** file
- **/usr/share/doc/rhel-system-roles/podman/** directory

CHAPTER 20. CONFIGURING POSTFIX MTA BY USING RHEL SYSTEM ROLES

You can use the **postfix** RHEL system role to consistently manage configurations of the Postfix mail transfer agent (MTA) in an automated fashion. Deploying such configurations are helpful when you need for example:

- Stable mail server: enables system administrators to configure a fast and scalable server for sending and receiving emails.
- Secure communication: supports features such as TLS encryption, authentication, domain blacklisting, and more, to ensure safe email transmission.
- Improved email management and routing: implements filters and rules so that you have control over your email traffic.

> **IMPORTANT**
>
> The **postfix_conf** dictionary holds key-value pairs of the supported Postfix configuration parameters. Those keys that Postfix does not recognize as supported are ignored. The **postfix** RHEL system role directly passes the key-value pairs that you provide to the **postfix_conf** dictionary without verifying their syntax or limiting them. Therefore, the role is especially useful to those familiar with Postfix, and who know how to configure it.

20.1. CONFIGURING POSTFIX AS A NULL CLIENT FOR ONLY SENDING OUTGOING EMAILS

A null client is a special configuration, where the Postfix server is set up only to send outgoing emails, but not receive any incoming emails. Such a setup is widely used in scenarios where you need to send notifications, alerts, or logs; but receiving or managing emails is not needed. By using Ansible and the **postfix** RHEL system role, you can automate this process and remotely configure the Postfix server as a null client for only sending outgoing emails.

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.

Procedure

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

```
---
- name: Manage Postfix
  hosts: managed-node-01.example.com
  tasks:
    - name: Install postfix
      ansible.builtin.package:
        name: postfix
        state: present
```

```
  - name: Configure null client for only sending outgoing emails
    ansible.builtin.include_role:
      name: redhat.rhel_system_roles.postfix
    vars:
      postfix_conf:
        myhostname: server.example.com
        myorigin: "$mydomain"
        relayhost: smtp.example.com
        inet_interfaces: loopback-only
        mydestination: ""
        relay_domains: "{{ lookup('ansible.builtin.pipe', 'postconf -h default_database_type')
}}:/etc/postfix/relay_domains"
      postfix_files:
        - name: relay_domains
          postmap: true
          content: |
            example.com OK
            example.net OK
```

The settings specified in the example playbook include the following:

myhostname: *<server.example.com>*

The internet hostname of this mail system. Defaults to the fully-qualified domain name (FQDN).

myorigin: $mydomain

The domain name that locally-posted mail appears to come from and that locally posted mail is delivered to. Defaults to **$myhostname**.

relayhost: *<smtp.example.com>*

The next-hop destination(s) for non-local mail, overrides non-local domains in recipient addresses. Defaults to an empty field.

inet_interfaces: loopback-only

Defines which network interfaces the Postfix server listens on for incoming email connections. It controls whether and how the Postfix server accepts email from the network.

mydestination

Defines which domains and hostnames are considered local.

relay_domains: "hash:/etc/postfix/relay_domains"

Specifies the domains that Postfix can forward emails to when it is acting as a relay server (SMTP relay). In this case the domains will be generated by the **postfix_files** variable. On RHEL 10, you have to use **relay_domains: "lmdb:/etc/postfix/relay_domains"**.

postfix_files

Defines a list of files that will be placed in the **/etc/postfix/** directory. Those files can be converted into Postfix Lookup Tables if needed. In this case **postfix_files** generates domain names for the SMTP relay.

For details about the role variables and the Postfix configuration parameters used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.postfix/README.md** file and the **postconf(5)** manual page on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.postfix/README.md** file
- **/usr/share/doc/rhel-system-roles/postfix/** directory
- **postconf(5)** manual page on your system

CHAPTER 21. INSTALLING AND CONFIGURING A POSTGRESQL DATABASE SERVER BY USING RHEL SYSTEM ROLES

You can use the **postgresql** RHEL system role to automate the installation and management of the PostgreSQL database server. By default, this role also optimizes PostgreSQL by automatically configuring performance-related settings in the PostgreSQL service configuration files.

21.1. CONFIGURING POSTGRESQL WITH AN EXISTING TLS CERTIFICATE BY USING THE POSTGRESQL RHEL SYSTEM ROLE

If your application requires a PostgreSQL database server, you can configure this service with TLS encryption to enable secure communication between the application and the database. By using the **postgresql** RHEL system role, you can automate this process and remotely install and configure PostgreSQL with TLS encryption. In the playbook, you can use an existing private key and a TLS certificate that was issued by a certificate authority (CA).

> **NOTE**
>
> The **postgresql** role cannot open ports in the **firewalld** service. To allow remote access to the PostgreSQL server, add a task that uses the **firewall** RHEL system role to your playbook.

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.
- Both the private key of the managed node and the certificate are stored on the control node in the following files:
  - Private key: ~/***<FQDN_of_the_managed_node>*.key**
  - Certificate: ~/***<FQDN_of_the_managed_node>*.crt**

Procedure

1. Store your sensitive variables in an encrypted file:
   a. Create the vault:

```
$ ansible-vault create ~/vault.yml
New Vault password: <vault_password>
Confirm New Vault password: <vault_password>
```

   b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

```
pwd: <password>
```

c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

```
---
- name: Installing and configuring PostgreSQL
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Create directory for TLS certificate and key
      ansible.builtin.file:
        path: /etc/postgresql/
        state: directory
        mode: 755

    - name: Copy CA certificate
      ansible.builtin.copy:
        src: "~/{{ inventory_hostname }}.crt"
        dest: "/etc/postgresql/server.crt"

    - name: Copy private key
      ansible.builtin.copy:
        src: "~/{{ inventory_hostname }}.key"
        dest: "/etc/postgresql/server.key"
        mode: 0600

    - name: PostgreSQL with an existing private key and certificate
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.postgresql
      vars:
        postgresql_version: "16"
        postgresql_password: "{{ pwd }}"
        postgresql_ssl_enable: true
        postgresql_cert_name: "/etc/postgresql/server"
        postgresql_server_conf:
          listen_addresses: "'*'"
          password_encryption: scram-sha-256
        postgresql_pg_hba_conf:
          - type: local
            database: all
            user: all
            auth_method: scram-sha-256
          - type: hostssl
            database: all
            user: all
            address: '127.0.0.1/32'
            auth_method: scram-sha-256
          - type: hostssl
            database: all
            user: all
            address: '::1/128'
            auth_method: scram-sha-256
          - type: hostssl
            database: all
            user: all
```

```
        address: '192.0.2.0/24'
        auth_method: scram-sha-256


  - name: Open the PostgresQL port in firewalld
    ansible.builtin.include_role:
      name: redhat.rhel_system_roles.firewall
    vars:
      firewall:
        - service: postgresql
          state: enabled
```

The settings specified in the example playbook include the following:

postgresql_version: *<version>*

Sets the version of PostgreSQL to install. The version you can set depends on the PostgreSQL versions that are available in Red Hat Enterprise Linux running on the managed node.
You cannot upgrade or downgrade PostgreSQL by changing the **postgresql_version** variable and running the playbook again.

postgresql_password: *<password>*

Sets the password of the **postgres** database superuser.
You cannot change the password by changing the **postgresql_password** variable and running the playbook again.

postgresql_cert_name: *<private_key_and_certificate_file>*

Defines the path and base name of both the certificate and private key on the managed node without **.crt** and **key** suffixes. During the PostgreSQL configuration, the role creates symbolic links in the **/var/lib/pgsql/data/** directory that refer to these files.
The certificate and private key must exist locally on the managed node. You can use tasks with the **ansible.builtin.copy** module to transfer the files from the control node to the managed node, as shown in the playbook.

postgresql_server_conf: *<list_of_settings>*

Defines **postgresql.conf** settings the role should set. The role adds these settings to the **/etc/postgresql/system-roles.conf** file and includes this file at the end of **/var/lib/pgsql/data/postgresql.conf**. Consequently, settings from the **postgresql_server_conf** variable override settings in **/var/lib/pgsql/data/postgresql.conf**.
Re-running the playbook with different settings in **postgresql_server_conf** overwrites the **/etc/postgresql/system-roles.conf** file with the new settings.

postgresql_pg_hba_conf: *<list_of_authentication_entries>*

Configures client authentication entries in the **/var/lib/pgsql/data/pg_hba.conf** file. For details, see see the PostgreSQL documentation.
The example allows the following connections to PostgreSQL:

- Unencrypted connections by using local UNIX domain sockets.
- TLS-encrypted connections to the IPv4 and IPv6 localhost addresses.

- TLS-encrypted connections from the 192.0.2.0/24 subnet. Note that access from remote addresses is only possible if you also configure the **listen_addresses** setting in the **postgresql_server_conf** variable appropriately.

Re-running the playbook with different settings in **postgresql_pg_hba_conf** overwrites the **/var/lib/pgsql/data/pg_hba.conf** file with the new settings.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.postgresql/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --ask-vault-pass --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Verification

- Use the **postgres** super user to connect to a PostgreSQL server and execute the **\conninfo** meta command:

```
# psql "postgresql://postgres@managed-node-01.example.com:5432" -c '\conninfo'
Password for user postgres:
You are connected to database "postgres" as user "postgres" on host "192.0.2.1" at port
"5432".
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression:
off)
```

If the output displays a TLS protocol version and cipher details, the connection works and TLS encryption is enabled.

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.postgresql/README.md** file
- **/usr/share/doc/rhel-system-roles/postgresql/** directory
- Ansible vault

21.2. CONFIGURING POSTGRESQL WITH A TLS CERTIFICATE ISSUED FROM IDM BY USING THE POSTGRESQL RHEL SYSTEM ROLE

If your application requires a PostgreSQL database server, you can configure the PostgreSQL service with TLS encryption to enable secure communication between the application and the database. If the PostgreSQL host is a member of a Red Hat Enterprise Linux Identity Management (IdM) domain, the **certmonger** service can manage the certificate request and future renewals.

By using the **postgresql** RHEL system role, you can automate this process. You can remotely install and configure PostgreSQL with TLS encryption, and the **postgresql** role uses the **certificate** RHEL system role to configure **certmonger** and request a certificate from IdM.

> **NOTE**
>
> The **postgresql** role cannot open ports in the **firewalld** service. To allow remote access to the PostgreSQL server, add a task to your playbook that uses the **firewall** RHEL system role.

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.
- You enrolled the managed node in an IdM domain.

Procedure

1. Store your sensitive variables in an encrypted file:

   a. Create the vault:

   ```
   $ ansible-vault create ~/vault.yml
   New Vault password: <vault_password>
   Confirm New Vault password: <vault_password>
   ```

   b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

   ```
   pwd: <password>
   ```

   c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

```
---
- name: Installing and configuring PostgreSQL
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: PostgreSQL with certificates issued by IdM
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.postgresql
      vars:
        postgresql_version: "16"
        postgresql_password: "{{ pwd }}"
        postgresql_ssl_enable: true
        postgresql_certificates:
          - name: postgresql_cert
            dns: "{{ inventory_hostname }}"
```

```
          ca: ipa
          principal: "postgresql/{{ inventory_hostname }}@EXAMPLE.COM"
        postgresql_server_conf:
          listen_addresses: "'*'"
          password_encryption: scram-sha-256
        postgresql_pg_hba_conf:
          - type: local
            database: all
            user: all
            auth_method: scram-sha-256
          - type: hostssl
            database: all
            user: all
            address: '127.0.0.1/32'
            auth_method: scram-sha-256
          - type: hostssl
            database: all
            user: all
            address: '::1/128'
            auth_method: scram-sha-256
          - type: hostssl
            database: all
            user: all
            address: '192.0.2.0/24'
            auth_method: scram-sha-256


    - name: Open the PostgresQL port in firewalld
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.firewall
      vars:
        firewall:
          - service: postgresql
            state: enabled
```

The settings specified in the example playbook include the following:

postgresql_version: *<version>*

Sets the version of PostgreSQL to install. The version you can set depends on the PostgreSQL versions that are available in Red Hat Enterprise Linux running on the managed node.
You cannot upgrade or downgrade PostgreSQL by changing the **postgresql_version** variable and running the playbook again.

postgresql_password: *<password>*

Sets the password of the **postgres** database superuser.
You cannot change the password by changing the **postgresql_password** variable and running the playbook again.

postgresql_certificates: *<certificate_role_settings>*

A list of YAML dictionaries with settings for the **certificate** role.

postgresql_server_conf: *<list_of_settings>*

Defines **postgresql.conf** settings you want the role to set. The role adds these settings to the **/etc/postgresql/system-roles.conf** file and includes this file at the end of

/var/lib/pgsql/data/postgresql.conf. Consequently, settings from the **postgresql_server_conf** variable override settings in **/var/lib/pgsql/data/postgresql.conf**. Re-running the playbook with different settings in **postgresql_server_conf** overwrites the **/etc/postgresql/system-roles.conf** file with the new settings.

postgresql_pg_hba_conf: *<list_of_authentication_entries>*

Configures client authentication entries in the **/var/lib/pgsql/data/pg_hba.conf** file. For details, see see the PostgreSQL documentation.
The example allows the following connections to PostgreSQL:

- Unencrypted connections by using local UNIX domain sockets.
- TLS-encrypted connections to the IPv4 and IPv6 localhost addresses.
- TLS-encrypted connections from the 192.0.2.0/24 subnet. Note that access from remote addresses is only possible if you also configure the **listen_addresses** setting in the **postgresql_server_conf** variable appropriately.

Re-running the playbook with different settings in **postgresql_pg_hba_conf** overwrites the **/var/lib/pgsql/data/pg_hba.conf** file with the new settings.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.postgresql/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --ask-vault-pass --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Verification

- Use the **postgres** super user to connect to a PostgreSQL server and execute the **\conninfo** meta command:

```
# psql "postgresql://postgres@managed-node-01.example.com:5432" -c '\conninfo'
Password for user postgres:
You are connected to database "postgres" as user "postgres" on host "192.0.2.1" at port
"5432".
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression:
off)
```

If the output displays a TLS protocol version and cipher details, the connection works and TLS encryption is enabled.

Additional resources

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

- **/usr/share/doc/rhel-system-roles/postgresql/** directory
- Ansible vault

CHAPTER 22. REGISTERING THE SYSTEM BY USING RHEL SYSTEM ROLES

The **rhc** RHEL system role enables administrators to automate the registration of multiple systems with Red Hat Subscription Management (RHSM) and Satellite servers. The role also supports Insights-related configuration and management tasks by using Ansible. By default, when you register a system by using **rhc**, the system is connected to Red Hat Insights. Additionally, with **rhc**, you can:

- Configure connections to Red Hat Insights
- Enable and disable repositories
- Configure the proxy to use for the connection
- Configure Insights remediations and, auto updates
- Set the release of the system
- Configure Insights tags

22.1. REGISTERING A SYSTEM BY USING THE RHC RHEL SYSTEM ROLE

You can register multiple systems at scale with Red Hat subscription management (RHSM) by using the **rhc** RHEL system role. By default, **rhc** connects the system to Red Hat Insights when you register it. Registering your system enables features and capabilities that you can use to manage your system and report data.

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.

Procedure

1. Store your sensitive variables in an encrypted file:

   a. Create the vault:

   ```
   $ ansible-vault create ~/vault.yml
   New Vault password: <password>
   Confirm New Vault password: <vault_password>
   ```

   b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

   ```
   activationKey: <activation_key>
   organizationID: <organizationID>
   username: <username>
   password: <password>
   ```

   c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

   - To register by using an activation key and organization ID (recommended), use the following playbook:

     ```
     ---
     - name: Managing systems with the rhc RHEL system role
       hosts: managed-node-01.example.com
       vars_files:
         - ~/vault.yml
       tasks:
         - name: Registering system using activation key and organization ID
           ansible.builtin.include_role:
             name: redhat.rhel_system_roles.rhc
           vars:
             rhc_auth:
               activation_keys:
                 keys:
                   - "{{ activationKey }}"
             rhc_organization: "{{ organizationID }}"
     ```

     The settings specified in the example playbook include the following:

     rhc_auth: ***activation_keys***
     The key **activation_keys** specifies that you want to register by using the activation keys.

   - To register by using a username and password, use the following playbook:

     ```
     ---
     - name: Managing systems with the rhc RHEL system role
       hosts: managed-node-01.example.com
       vars_files:
         - ~/vault.yml
       tasks:
         - name: Registering system with username and password
           ansible.builtin.include_role:
             name: redhat.rhel_system_roles.rhc
           vars:
             rhc_auth:
               login:
                 username: "{{ username }}"
                 password: "{{ password }}"
     ```

   The settings specified in the example playbook include the following:

   rhc_auth: ***login***
   The key **login** specifies that you want to register by using the username and password.

3. Validate the playbook syntax:

   ```
   $ ansible-playbook --syntax-check --ask-vault-pass ~/playbook.yml
   ```

   Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.rhc/README.md** file
- **/usr/share/doc/rhel-system-roles/rhc/** directory
- Ansible Vault

22.2. REGISTERING A SYSTEM WITH SATELLITE BY USING THE RHC RHEL SYSTEM ROLE

When organizations use Satellite to manage systems, it is necessary to register the system through Satellite. You can remotely register your system with Satellite by using the **rhc** RHEL system role.

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.

Procedure

1. Store your sensitive variables in an encrypted file:

   a. Create the vault:

   ```
   $ ansible-vault create ~/vault.yml
   New Vault password: <password>
   Confirm New Vault password: <vault_password>
   ```

   b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

   ```
   activationKey: <activation_key>
   organizationID: <organizationID>
   ```

   c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

```
---
- name: Managing systems with the rhc RHEL system role
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Register to the custom registration server and CDN
```

```
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.rhc
      vars:
        rhc_auth:
          activation_keys:
            keys:
              - "{{ activationKey }}"
        rhc_organization: "{{ organizationID }}"
        rhc_server:
          hostname: example.com
            port: 443
            prefix: /rhsm
        rhc_baseurl: http://example.com/pulp/content
```

The settings specified in the example playbook include the following:

hostname: *example.com*
: A fully qualified domain name (FQDN) of the Satellite server for system registration and package management.

port: *443*
: Defines the network port used for communication with the Satellite server.

prefix: */rhsm*
: Specifies the URL path prefix for accessing resources on the Satellite server.

rhc_baseurl: *http://example.com/pulp/content*
: Defines the prefix for content URLs. In a Satellite environment, the **baseurl** must be set to the same server where the system is registered. Refer to the **hostname** value to ensure the correct server is used.

3. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check --ask-vault-pass ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.rhc/README.md** file
- **/usr/share/doc/rhel-system-roles/rhc/** directory
- Ansible Vault

22.3. DISABLING THE CONNECTION TO INSIGHTS AFTER THE REGISTRATION BY USING THE RHC RHEL SYSTEM ROLE

When you register a system by using the **rhc** RHEL system role, the role by default, enables the connection to Red Hat Insights. Red Hat Insights is a managed service in the Hybrid Cloud Console that uses predictive analytics, remediation capabilities, and deep domain expertise to simplify complex operational tasks. You can disable it by using the **rhc** RHEL system role, if not required.

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.
- You have registered the system.

Procedure

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

```
---
- name: Managing systems with the rhc RHEL system role
  hosts: managed-node-01.example.com
  tasks:
    - name: Disable Insights connection
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.rhc
      vars:
        rhc_insights:
          state: absent
```

   The settings specified in the example playbook include the following:

   rhc_insights *absent|present*
   Enables or disables system registration with Red Hat Insights for proactive analytics and recommendations.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.rhc/README.md** file
- **/usr/share/doc/rhel-system-roles/rhc/** directory

22.4. MANAGING REPOSITORIES BY USING THE RHC RHEL SYSTEM ROLE

Enabling repositories on a RHEL system is essential for accessing, installing, and updating software packages from verified sources. You can remotely enable or disable repositories on managed nodes by using **rhc** RHEL system role to ensure the system security, stability, and compatibility.

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.
- You have details of the repositories which you want to enable or disable on the managed nodes.
- You have registered the system.

Procedure

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

```
---
- name: Managing systems with the rhc RHEL system role
  hosts: managed-node-01.example.com
  tasks:
    - name: Enable repository
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.rhc
      vars:
        rhc_repositories:
          - name: "RepositoryName"
            state: enabled
```

The settings specified in the example playbook include the following:

name: *RepositoryName*
: Name of the repository that should be enabled.

state: *enabled|disabled*
: Optional, enables or disables the repository. Default is **enabled**.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Additional resources

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

- **/usr/share/doc/rhel-system-roles/rhc/** directory

22.5. LOCKING THE SYSTEM TO A PARTICULAR RELEASE BY USING THE RHC RHEL SYSTEM ROLE

To ensure system stability and compatibility, it is sometimes necessary to limit the RHEL system to use only repositories from a specific minor version rather than automatically upgrading to the latest available release. Locking the system to a particular minor version helps maintain consistency in production environments, which prevents unintended updates that might introduce compatibility issues.

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.
- You know the RHEL version to which you want to lock the system. Note that you can only lock the system to the RHEL minor version that the managed node currently runs or a later minor version.
- You have registered the system.

Procedure

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

```
---
- name: Managing systems with the rhc RHEL system role
  hosts: managed-node-01.example.com
  tasks:
    - name: Lock the system to a particular release
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.rhc
      vars:
        rhc_release: "8.6"
```

The settings specified in the example playbook include the following:

rhc_release: *version*
The version of RHEL to set for the system, so the available content will be limited to that version.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.rhc/README.md** file
- **/usr/share/doc/rhel-system-roles/rhc/** directory
- Red Hat Enterprise Linux (RHEL) Extended Update Support (EUS) Overview (Red Hat Knowledgebase).

22.6. USING A PROXY SERVER WHEN REGISTERING THE HOST BY USING THE RHC RHEL SYSTEM ROLE

If your security restrictions allow access to the Internet only through a proxy server, you can specify the proxy settings of the **rhc** role when you register the system using **rhc**.

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.

Procedure

1. Store your sensitive variables in an encrypted file:
    a. Create the vault:

    ```
    $ ansible-vault create ~/vault.yml
    New Vault password: <password>
    Confirm New Vault password: <vault_password>
    ```

    b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

    ```
    username: <username>
    password: <password>
    proxy_username: <proxyusernme>
    proxy_password: <proxypassword>
    ```

    c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

```
---
- name: Managing systems with the rhc RHEL system role
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
  - name: Register to the Red Hat Customer Portal by using proxy
    ansible.builtin.include_role:
      name: redhat.rhel_system_roles.rhc
```

```
    vars:
      rhc_auth:
        login:
          username: "{{ username }}"
          password: "{{ password }}"
      rhc_proxy:
        hostname: proxy.example.com
        port: 3128
        username: "{{ proxy_username }}"
        password: "{{ proxy_password }}"
```

The settings specified in the example playbook include the following:

hostname: *proxy.example.com*
: A fully qualified domain name (FQDN) of the proxy server.

port: *3128*
: Defines the network port used for communication with the proxy server.

username: *proxy_username*
: Specifies the username for authentication. This is required only if the proxy server requires authentication.

password: *proxy_password*
: Specifies the password for authentication. This is required only if the proxy server requires authentication.

3. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check --ask-vault-pass ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.rhc/README.md** file
- **/usr/share/doc/rhel-system-roles/rhc/** directory
- Ansible Vault

22.7. MANAGING AUTO UPDATES OF INSIGHTS RULES BY USING THE RHC RHEL SYSTEM ROLE

You can enable or disable the automatic collection rule updates for Red Hat Insights by using the **rhc** RHEL system role. By default, when you connect your system to Red Hat Insights, this option is enabled. You can disable it by using **rhc**.

> **WARNING**
>
> If you disable this feature, you risk using outdated rule definition files and not getting the most recent validation updates.

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.
- You have registered the system.

Procedure

1. Store your sensitive variables in an encrypted file:

   a. Create the vault:

   ```
   $ ansible-vault create ~/vault.yml
   New Vault password: <password>
   Confirm New Vault password: <vault_password>
   ```

   b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

   ```
   username: <username>
   password: <password>
   ```

   c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

```
---
- name: Managing systems with the rhc RHEL system role
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Enable Red Hat Insights autoupdates
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.rhc
      vars:
        rhc_auth:
          login:
            username: "{{ username }}"
            password: "{{ password }}"
```

```
      rhc_insights:
        autoupdate: true
        state: present
```

The settings specified in the example playbook include the following:

autoupdate: *true*/*false*
: Enables or disables the automatic collection rule updates for Red Hat Insights.

3. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check --ask-vault-pass ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.rhc/README.md** file
- **/usr/share/doc/rhel-system-roles/rhc/** directory
- Ansible Vault

22.8. CONFIGURING INSIGHTS REMEDIATIONS BY USING THE RHC RHEL SYSTEM ROLE

You can configure your systems to automatically update the dynamic configuration by using the **rhc** RHEL system role. When you connect your system to Red Hat Insights, it is enabled by default. You can disable it, if not required. You can use **rhc** to ensure your system is ready for remediation when connected directly to Red Hat. For more information about Red Hat Insights remediations, see Red Hat Insights Remediations Guide.

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.
- You have Insights remediations enabled.
- You have registered the system.

Procedure

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

```
---
```

```
- name: Managing systems with the rhc RHEL system role
  hosts: managed-node-01.example.com
  tasks:
    - name: Disable remediation
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.rhc
      vars:
        rhc_insights:
          remediation: absent
          state: present
```

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.rhc/README.md** file
- **/usr/share/doc/rhel-system-roles/rhc/** directory

22.9. CONFIGURING INSIGHTS TAGS BY USING THE RHC RHEL SYSTEM ROLE

You can use the **rhc** RHEL system role to configure Red Hat Insights tags for system filtering and grouping. You can also customize tags based on the requirements. Filtering and grouping systems by using Red Hat Insights tags help administrators efficiently manage, monitor, and apply policies to specific sets of systems based on attributes like environment, location, or function. This improves visibility, simplifies automation, and enhances security compliance across large infrastructures.

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.

Procedure

1. Store your sensitive variables in an encrypted file:
   a. Create the vault:

```
$ ansible-vault create ~/vault.yml
New Vault password: <password>
Confirm New Vault password: <vault_password>
```

b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

```
username: <username>
password: <password>
```

c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

```
---
- name: Managing systems with the rhc RHEL system role
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Creating tags
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.rhc
      vars:
        rhc_auth:
          login:
            username: "{{ username }}"
            password: "{{ password }}"
        rhc_insights:
          tags:
            group: group-name-value
            location: location-name-value
            description:
              - RHEL8
              - SAP
            sample_key: value
          state: present
```

The settings specified in the example playbook include the following:

group: ***group-name-value***
: Specifies the system group for organizing and managing registered hosts.

location: ***location-name-value***
: Defines the location associated with the registered system.

description
: Provides a brief summary or identifier for the registered system.

state: ***present|absent***
: Indicates the current status of the registered system.

> **NOTE**
>
> The content inside the **tags** is a YAML structure representing the tags desired by the administrator for the configured systems. The example provided here is for illustrative purposes only and is not exhaustive. Administrators can customize the YAML structure to include any additional keys and values as needed.

3. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check --ask-vault-pass ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.rhc/README.md** file
- **/usr/share/doc/rhel-system-roles/rhc/** directory
- Ansible Vault
- Custom system tagging

22.10. UNREGISTERING A SYSTEM BY USING THE RHC RHEL SYSTEM ROLE

You can use the **rhc** RHEL system role to unregister the system from the Red Hat subscription service if you no longer want to receive content from the registration server on a specific system, for example, system decommissioning, VM deletion, or when switching to a local content mirror.

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 system is already registered.

Procedure

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

```
---
- name: Managing systems with the rhc RHEL system role
  hosts: managed-node-01.example.com
  tasks:
    - name: Unregister the system
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.rhc
      vars:
        rhc_state: absent
```

The settings specified in the example playbook include the following:

rhc_state: *absent*

Specifies the system should be unregistered from the registration server, RHSM, or Satellite.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.rhc/README.md** file
- **/usr/share/doc/rhel-system-roles/rhc/** directory

CHAPTER 23. REMOTE MANAGEMENT WITH IPMI AND REDFISH BY USING THE RHEL_MGMT COLLECTION

With the Intelligent Platform Management Interface (IPMI) and the Redfish API, administrators can remotely manage hosts even if the operating system is not running. The **rhel_mgmt** Ansible collection provides modules that use IPMI and Redfish to perform certain remote operations, such as setting the boot device.

23.1. SETTING THE BOOT DEVICE BY USING THERHEL_MGMT.IPMI_BOOT MODULE

You can set the boot device of a host by using the **ipmi_boot** module of the **redhat.rhel_mgmt** collection. This module uses the Intelligent Platform Management Interface (IPMI) to perform this operation.

IMPORTANT

When you use this Ansible module, three hosts are involved: the control node, the managed node, and the host with the baseboard management controller (BMC) on which the actual IPMI operation is applied. The control node executes the playbook on the managed node. The managed host connects to the remote BMC to execute the IPMI operation. For example, if you set **hosts: managed-node-01.example.com** and **name: server.example.com** in the playbook, then **managed-node-01.example.com** changes the setting by using IPMI on **server.example.com**.

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 **ansible-collection-redhat-rhel_mgmt** package is installed on the control node.
- You have credentials to access the BMC, and these credentials have permissions to change settings.
- The managed node can access the remote BMC over the network.

Procedure

1. Store your sensitive variables in an encrypted file:
   a. Create the vault:

   ```
   $ ansible-vault create ~/vault.yml
   New Vault password: <vault_password>
   Confirm New Vault password: <vault_password>
   ```

   b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

```
ipmi_usr: <username>
ipmi_pwd: <password>
```

c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

```
---
- name: Set the boot device by using IPMI
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Install python3-pyghmi prerequisite
      ansible.builtin.dnf:
        name: python3-pyghmi
        state: latest

    - name: Set the boot device to hd
      redhat.rhel_mgmt.ipmi_boot:
        name: <bmc_hostname_or_ip_address>
        port: <bmc_port_number>
        user: "{{ ipmi_usr }}"
        password: "{{ ipmi_pwd }}"
        bootdev: hd
        persistent: true
```

The settings specified in the example playbook include the following:

name: *<bmc_hostname_or_ip_address>*

Defines the hostname or IP address of the BMC. This is the BMC of the host on which the managed node performs the action.

port: *<bmc_port_number>*

Sets the Remote Management Control Protocol (RMCP) port number. The default is **623**.

bootdev: *<value>*

Sets the boot device. You can select one of the following values:

- **hd**: Boots from the hard disk.
- **network**: Boots from network.
- **optical**: Boots from an optical drive, such as a DVD-ROM.
- **floppy**: Boots from a floppy disk.
- **safe**: Boots from hard drive in safe mode.
- **setup**: Boots into the BIOS or UEFI.
- **default**: Removes any IPMI-directed boot device request.

persistent: *<true/false>*

Configures whether the remote host uses the defined setting for all future boots or only for the next one. By default, this variable is set to **false**. Note that not all BMCs support setting the boot device persistently.

For details about all variables used in the playbook, use the **ansible-doc redhat.rhel_mgmt.ipmi_boot** command on the control node to display the documentation of the module.

3. Validate the playbook syntax:

```
$ ansible-playbook --ask-vault-pass --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Additional resources

- The **ansible-doc redhat.rhel_mgmt.ipmi_boot** command on the control node
- Ansible vault

23.2. SETTING THE SYSTEM POWER STATE BY USING THE RHEL_MGMT.IPMI_POWER MODULE

You can set the hardware state by using the **ipmi_power** module of the **redhat.rhel_mgmt** collection. For example, you can ensure that a host is powered on or hard-reset it without involvement of the operating system. The **ipmi_power** module uses the Intelligent Platform Management Interface (IPMI) to perform operations.

> **IMPORTANT**
>
> When you use this Ansible module, three hosts are involved: the control node, the managed node, and the host with the baseboard management controller (BMC) on which the actual IPMI operation is applied. The control node executes the playbook on the managed node. The managed host connects to the remote BMC to execute the IPMI operation. For example, if you set **hosts: managed-node-01.example.com** and **name: server.example.com** in the playbook, then **managed-node-01.example.com** changes the setting by using IPMI on **server.example.com**.

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 **ansible-collection-redhat-rhel_mgmt** package is installed on the control node.

- You have credentials to access the BMC, and these credentials have permissions to change settings.
- The managed node can access the remote BMC over the network.

Procedure

1. Store your sensitive variables in an encrypted file:
   a. Create the vault:

   ```
   $ ansible-vault create ~/vault.yml
   New Vault password: <vault_password>
   Confirm New Vault password: <vault_password>
   ```

   b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

   ```
   ipmi_usr: <username>
   ipmi_pwd: <password>
   ```

   c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

```
---
- name: Configure the system state by using IPMI
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Install python3-pyghmi prerequisite
      ansible.builtin.dnf:
        name: python3-pyghmi
        state: latest

    - name: Ensure a machine is powered on
      redhat.rhel_mgmt.ipmi_power:
        name: <bmc_hostname_or_ip_address>
        port: <bmc_port_number>
        user: "{{ ipmi_usr }}"
        password: "{{ ipmi_pwd }}"
        state: on
```

The settings specified in the example playbook include the following:

name: *<bmc_hostname_or_ip_address>*
: Defines the hostname or IP address of the BMC. This is the BMC of the host on which the managed node performs the action.

port: *<bmc_port_number>*
: Sets the Remote Management Control Protocol (RMCP) port number. The default is **623**.

state: *<value>*
: Sets the state which the device should be in. You can select one of the following values:

- **on**: Powers on the system.
- **off**: Powers off the system without notifying the operating system.
- **shutdown**: Requests a shutdown from the operating system.
- **reset**: Performs a hard reset.
- **boot**: Powers on the system if it was switched off, or resets the system if it was switched off.

For details about all variables used in the playbook, use the **ansible-doc redhat.rhel_mgmt.ipmi_power** command on the control node to display the documentation of the module.

3. Validate the playbook syntax:

```
$ ansible-playbook --ask-vault-pass --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Additional resources

- The **ansible-doc redhat.rhel_mgmt.ipmi_power** command on the control node
- Ansible vault

23.3. MANAGING OUT-OF-BAND CONTROLLERS BY USING THE RHEL_MGMT.REDFISH_COMMAND MODULE

You can send commands to the Redfish API to remotely manage out-of-band (OOB) controllers by using the **redfish_command** module of the **redhat.rhel_mgmt** collection. With this module, you can perform a large number of management operations, for example:

- Performing power management actions
- Managing virtual media
- Managing users of the OOB controller
- Updating the firmware

IMPORTANT

When you use this Ansible module, three hosts are involved: the control node, the managed node, and the host with the OOB controller on which the actual operation is performed. The control node executes the playbook on the managed node, and the managed host connects to the remote OOB controller by using the Redfish API to execute the operation. For example, if you set **hosts: managed-node-01.example.com** and **baseuri: server.example.com** in the playbook, then **managed-node-01.example.com** executes the operation on **server.example.com**.

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 **ansible-collection-redhat-rhel_mgmt** package is installed on the control node.
- You have credentials to access the OOB controller, and these credentials have permissions to change settings.
- The managed node can access the remote OOB controller over the network.

Procedure

1. Store your sensitive variables in an encrypted file:

   a. Create the vault:

   ```
   $ ansible-vault create ~/vault.yml
   New Vault password: <vault_password>
   Confirm New Vault password: <vault_password>
   ```

   b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

   ```
   redfish_usr: <username>
   redfish_pwd: <password>
   ```

   c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

```
---
- name: Send commands to OOB controller by using the Redfish API
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Power on the system
      redhat.rhel_mgmt.redfish_command:
        baseuri: <uri>
        username: "{{ redfish_usr }}"
```

```
        password: "{{ redfish_pwd }}"
        category: Systems
        command: PowerOn
```

The settings specified in the example playbook include the following:

baseuri: *<uri>*
Defines the URI of the OOB controller. This is the OOB controller of the host on which the managed node performs the action.

category: *<value>*
Sets the category of the command to execute. The following categories are available:

- **Accounts**: Manages user accounts of the OOB controller.
- **Chassis**: Manages chassis-related settings.
- **Manager**: Provides access to Redfish services.
- **Session**: Manages Redfish login sessions.
- **Systems** (default): Manages machine-related settings.
- **Update**: Manages firmware update-related actions.

command: *<command>*
Sets the command to execute. Depending on the command, it can be necessary to set additional variables.

For details about all variables used in the playbook, use the **ansible-doc redhat.rhel_mgmt.redfish_command** command on the control node to display the documentation of the module.

3. Validate the playbook syntax:

```
$ ansible-playbook --ask-vault-pass --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Additional resources

- The **ansible-doc redhat.rhel_mgmt.ipmi_command** command on the control node
- Ansible vault

23.4. QUERYING INFORMATION FROM OUT-OF-BAND CONTROLLERS BY USING THE RHEL_MGMT.REDFISH_INFO MODULE

You can remotely query information from of out-of-band (OOB) controllers through the Redfish API by using the **redfish_info** module of the **redhat.rhel_mgmt** collection. To display the returned value, register a variable with the fetched information, and display the content of this variable.

> **IMPORTANT**
>
> When you use this Ansible module, three hosts are involved: the control node, the managed node, and the host with the OOB controller on which the actual operation is performed. The control node executes the playbook on the managed node, and the managed host connects to the remote OOB controller by using the Redfish API to execute the operation. For example, if you set **hosts: managed-node-01.example.com** and **baseuri: server.example.com** in the playbook, then **managed-node-01.example.com** executes the operation on **server.example.com**.

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 **ansible-collection-redhat-rhel_mgmt** package is installed on the control node.
- You have credentials to access the OOB controller, and these credentials have permissions to query settings.
- The managed node can access the remote OOB controller over the network.

Procedure

1. Store your sensitive variables in an encrypted file:
   a. Create the vault:

      ```
      $ ansible-vault create ~/vault.yml
      New Vault password: <vault_password>
      Confirm New Vault password: <vault_password>
      ```

   b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

      ```
      redfish_usr: <username>
      redfish_pwd: <password>
      ```

   c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

```
---
- name: Query information by using the Redfish API
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
```

```
- name: Get CPU inventory
  redhat.rhel_mgmt.redfish_info:
    baseuri: <uri>
    username: "{{ redfish_usr }}"
    password: "{{ redfish_pwd }}"
    category: Systems
    command: GetCpuInventory
  register: result

- name: Display the fetched information
  ansible.builtin.debug:
    msg: "{{ result.redfish_facts.cpu.entries | to_nice_json }}"
```

The settings specified in the example playbook include the following:

baseuri: *<uri>*

Defines the URI of the OOB controller. This is the OOB controller of the host on which the managed node performs the action.

category: *<value>*

Sets the category of the information to query. The following categories are available:

- **Accounts**: User accounts of the OOB controller
- **Chassis**: Chassis-related settings
- **Manager**: Redfish services
- **Session**: Redfish login sessions
- **Systems** (default): Machine-related settings
- **Update**: Firmware-related settings
- **All**: Information from all categories.

You can also set multiple categories if you use a list, for example **["Systems", "Accounts"]**.

command: *<command>*

Sets the query command to execute.

For details about all variables used in the playbook, use the **ansible-doc redhat.rhel_mgmt.redfish_info** command on the control node to display the documentation of the module.

3. Validate the playbook syntax:

```
$ ansible-playbook --ask-vault-pass --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Additional resources

- The **ansible-doc redhat.rhel_mgmt.redfish_info** command on the control node
- Ansible vault

23.5. MANAGING BIOS, UEFI, AND OUT-OF-BAND CONTROLLERS BY USING THE RHEL_MGMT.REDFISH_CONFIG MODULE

You can configure BIOS, UEFI, and out-of-band (OOB) controllers settings through the Redfish API by using the **redfish_config** module of the **redhat.rhel_mgmt** collection. This enables you to modify the settings remotely with Ansible.

> **IMPORTANT**
>
> When you use this Ansible module, three hosts are involved: the control node, the managed node, and the host with the OOB controller on which the actual operation is performed. The control node executes the playbook on the managed node, and the managed host connects to the remote OOB controller by using the Redfish API to execute the operation. For example, if you set **hosts: managed-node-01.example.com** and **baseuri: server.example.com** in the playbook, then **managed-node-01.example.com** executes the operation on **server.example.com**.

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 **ansible-collection-redhat-rhel_mgmt** package is installed on the control node.
- You have credentials to access the OOB controller, and these credentials have permissions to change settings.
- The managed node can access the remote OOB controller over the network.

Procedure

1. Store your sensitive variables in an encrypted file:

   a. Create the vault:

   ```
   $ ansible-vault create ~/vault.yml
   New Vault password: <vault_password>
   Confirm New Vault password: <vault_password>
   ```

   b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

   ```
   redfish_usr: <username>
   redfish_pwd: <password>
   ```

c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

```
---
- name: Configure BIOS/UEFI settings by using the Redfish API
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Change the boot mode to UEFI
      redhat.rhel_mgmt.redfish_config:
        baseuri: <uri>
        username: "{{ redfish_usr }}"
        password: "{{ redfish_pwd }}"
        category: Systems
        command: SetBiosAttributes
        bios_attributes:
          BootMode: "Uefi"
```

The settings specified in the example playbook include the following:

baseuri: *<uri>*
: Defines the URI of the OOB controller. This is the OOB controller of the host on which the managed node performs the action.

category: *<value>*
: Sets the category of the command to execute. The following categories are available:

- **Accounts**: Manages user accounts of the OOB controller.
- **Chassis**: Manages chassis-related settings.
- **Manager**: Provides access to Redfish services.
- **Session**: Manages Redfish login sessions.
- **Systems** (default): Manages machine-related settings.
- **Update**: Manages firmware update-related actions.

command: *<command>*
: Sets the command to execute. Depending on the command, it can be necessary to set additional variables.

For details about all variables used in the playbook, use the **ansible-doc redhat.rhel_mgmt.redfish_config** command on the control node to display the documentation of the module.

3. Validate the playbook syntax:

```
$ ansible-playbook --ask-vault-pass --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Additional resources

- The **ansible-doc redhat.rhel_mgmt.redfish_config** command on the control node
- Ansible vault

CHAPTER 24. CONFIGURING SELINUX BY USING RHEL SYSTEM ROLES

You can remotely configure and manage SELinux permissions by using the **selinux** RHEL system role, for example:

- 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.

24.1. RESTORING THE SELINUX CONTEXT ON DIRECTORIES BY USING THE SELINUX RHEL SYSTEM ROLE

There can be multiple cases when files have an incorrect SELinux context than. For example, if files are copied or moved to a directory, their SELinux context might not match the new location's expected context. With an incorrect SELinux context, applications might fail to access the files. To remotely reset the SELinux context on directories on a large number of hosts, you can use the **selinux** RHEL system role.

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.

Procedure

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

```
---
- name: Managing SELinux
  hosts: managed-node-01.example.com
  tasks:
    - name: Restore SELinux context
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.selinux
      vars:
        selinux_restore_dirs:
          - /var/www/
          - /etc/
```

The settings specified in the example playbook include the following:

selinux_restore_dirs: *<list>*
: Defines the list of directories on which the role should reset the SELinux context.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.selinux/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Display the SELinux context for files or directories for which you have reset the context. For example, to display the context on the **/var/www/** directory, enter:

```
# ansible rhel9.example.com -m command -a 'ls -ldZ /var/www/'
drwxr-xr-x. 4 root root system_u:object_r:httpd_sys_content_t:s0 33 Feb 28 13:20 /var/www/
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.selinux/README.md** file
- **/usr/share/doc/rhel-system-roles/selinux/** directory

24.2. MANAGING SELINUX NETWORK PORT LABELS BY USING THE SELINUX RHEL SYSTEM ROLE

If you want to run a service on a non-standard port, you must set the corresponding SELinux type label on this port. This prevents that SELinux denies permission to the service when the service wants to listen on the non-standard port. By using the **selinux** RHEL system role, you can automate this task and remotely assign a type label on ports.

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.

Procedure

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

```
---
- name: Managing SELinux
  hosts: managed-node-01.example.com
  tasks:
    - name: Set http_port_t label on network port
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.selinux
      vars:
```

```
      selinux_ports:
        - ports: <port_number>
          proto: tcp
          setype: http_port_t
          state: present
```

The settings specified in the example playbook include the following:

ports: *<port_number>*
: Defines the port numbers to which you want to assign the SELinux label. Separate multiple values by comma.

setype: *<type_label>*
: Defines the SELinux type label.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.selinux/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Display the port numbers that have the **http_port_t** label assigned:

```
# ansible managed-node-01.example.com -m shell -a 'semanage port --list | grep http_port_t'
http_port_t      tcp      80, 81, 443, <port_number>, 488, 8008, 8009, 8443, 9000
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.selinux/README.md** file
- **/usr/share/doc/rhel-system-roles/selinux/** directory

24.3. DEPLOYING AN SELINUX MODULE BY USING THE SELINUX RHEL SYSTEM ROLE

If the default SELinux policies do not meet your requirements, you can create custom modules to allow your application to access the required resources. By using the **selinux** RHEL system role, you can automate this process and remotely deploy SELinux modules.

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 SELinux module you want to deploy is stored in the same directory as the playbook.
- The SELinux module is available in the Common Intermediate Language (CIL) or policy package (PP) format.
  If you are using a PP module, ensure that **policydb** version on the managed nodes is the same or later than the version used to build the PP module.

Procedure

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

```
---
- name: Managing SELinux
  hosts: managed-node-01.example.com
  tasks:
    - name: Deploying a SELinux module
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.selinux
      vars:
        selinux_modules:
          - path: <module_file>
     priority: <value>
            state: enabled
```

The settings specified in the example playbook include the following:

path: *<module_file>*
: Sets the path to the module file on the control node.

priority: *<value>*
: Sets the SELinux module priority. **400** is the default.

state: *<value>*
: Defines the state of the module:

  - **enabled**: Install or enable the module.
  - **disabled**: Disable a module.
  - **absent**: Remove a module.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.selinux/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Remotely display the list of SELinux modules and filter for the one you used in the playbook:

```
# ansible managed-node-01.example.com -m shell -a 'semodule -l | grep <module>'
```

If the module is listed, it is installed and enabled.

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.selinux/README.md** file
- **/usr/share/doc/rhel-system-roles/selinux/** directory

CHAPTER 25. CONFIGURING THE OPENSSH SERVER AND CLIENT BY USING RHEL SYSTEM ROLES

You can use the **sshd** RHEL system role to configure OpenSSH servers and the **ssh** RHEL system role to configure OpenSSH clients consistently, in an automated fashion, and on any number of RHEL systems at the same time. Such configurations are necessary for any system where secure remote interaction is needed, for example:

- Remote system administration: securely connecting to your machine from another computer using an SSH client.
- Secure file transfers: the Secure File Transfer Protocol (SFTP) provided by OpenSSH enable you to securely transfer files between your local machine and a remote system.
- Automated DevOps pipelines: automating software deployments that require secure connection to remote servers (CI/CD pipelines).
- Tunneling and port forwarding: forwarding a local port to access a web service on a remote server behind a firewall. For example a remote database or a development server.
- Key-based authentication: more secure alternative to password-based logins.
- Certificate-based authentication: centralized trust management and better scalability.
- Enhanced security: disabling root logins, restricting user access, enforcing strong encryption and other such forms of hardening ensures stronger system security.

25.1. HOW THE SSHD RHEL SYSTEM ROLE MAPS SETTINGS FROM A PLAYBOOK TO THE CONFIGURATION FILE

In the **sshd** RHEL system role playbook, you can define the parameters for the server SSH configuration file.

If you do not specify these settings, the role produces the **sshd_config** file that matches the RHEL defaults.

In all cases, booleans correctly render as **yes** and **no** in the final configuration on your managed nodes. You can use lists to define multi-line configuration items. For example:

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

renders as:

```
ListenAddress 0.0.0.0
ListenAddress ::
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.sshd/README.md** file
- **/usr/share/doc/rhel-system-roles/sshd/** directory

25.2. CONFIGURING OPENSSH SERVERS BY USING THE SSHD RHEL SYSTEM ROLE

You can use the **sshd** RHEL system role to configure multiple OpenSSH servers. These ensure secure communication environment for remote users by providing namely:

- Management of incoming SSH connections from remote clients
- Credentials verification
- Secure data transfer and command execution

> **NOTE**
>
> You can use the **sshd** RHEL system role alongside with other RHEL system roles that change SSHD configuration, for example the Identity Management RHEL system roles. To prevent the configuration from being overwritten, ensure the **sshd** RHEL system role uses namespaces (RHEL 8 and earlier versions) or a drop-in directory (RHEL 9).

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.

Procedure

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

```
---
- name: SSH server configuration
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure sshd to prevent root and password login except from particular subnet
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.sshd
      vars:
        sshd:
          PermitRootLogin: no
          PasswordAuthentication: no
          Match:
            - Condition: "Address 192.0.2.0/24"
              PermitRootLogin: yes
              PasswordAuthentication: yes
```

The settings specified in the example playbook include the following:

PasswordAuthentication: yes|no
: Controls whether the OpenSSH server (**sshd**) accepts authentication from clients that use the username and password combination.

Match:
: The match block allows the **root** user login using password only from the subnet **192.0.2.0/24**.

For details about the role variables and the OpenSSH configuration options used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.sshd/README.md** file and the **sshd_config(5)** manual page on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

1. Log in to the SSH server:

```
$ ssh <username>@<ssh_server>
```

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

```
$ cat /etc/ssh/sshd_config
...
PasswordAuthentication no
PermitRootLogin no
...
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@<ssh_server>
   ```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.sshd/README.md** file
- **/usr/share/doc/rhel-system-roles/sshd/** directory

25.3. USING THE SSHD RHEL SYSTEM ROLE FOR NON-EXCLUSIVE CONFIGURATION

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

You can apply a non-exclusive configuration:

- In RHEL 8 and earlier by using a configuration snippet.
- In RHEL 9 and later by using files in a drop-in directory. The default configuration file is already placed in the drop-in directory as **/etc/ssh/sshd_config.d/00-ansible_system_role.conf**.

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.

Procedure

1. Create a playbook file, for example **~/playbook.yml**, with the following content:
   - For managed nodes that run RHEL 8 or earlier:

```
---
- name: Non-exclusive sshd configuration
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure SSHD to accept environment variables
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.sshd
      vars:
        sshd_config_namespace: <my-application>
        sshd:
          # Environment variables to accept
          AcceptEnv:
            LANG
            LS_COLORS
            EDITOR
```

   - For managed nodes that run RHEL 9 or later:

```
- name: Non-exclusive sshd configuration
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure sshd to accept environment variables
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.sshd
      vars:
        sshd_config_file: /etc/ssh/sshd_config.d/<42-my-application>.conf
```

```
    sshd:
      # Environment variables to accept
      AcceptEnv:
        LANG
        LS_COLORS
        EDITOR
```

The settings specified in the example playbooks include the following:

sshd_config_namespace: *<my-application>*
: The role places the configuration that you specify in the playbook to configuration snippets in the existing configuration file under the given namespace. You need to select a different namespace when running the role from different context.

sshd_config_file: /etc/ssh/sshd_config.d/*<42-my-application>*.conf
: In the **sshd_config_file** variable, define the **.conf** file into which the **sshd** system role writes the configuration options. Use a two-digit prefix, for example ***42-*** to specify the order in which the configuration files will be applied.

AcceptEnv:
: Controls which environment variables the OpenSSH server (**sshd**) will accept from a client:

  - **LANG**: defines the language and locale settings.
  - **LS_COLORS**: defines the displaying color scheme for the **ls** command in the terminal.
  - **EDITOR**: specifies the default text editor for the command-line programs that need to open an editor.

For details about the role variables and the OpenSSH configuration options used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.sshd/README.md** file and the **sshd_config(5)** manual page on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Verify the configuration on the SSH server:
  - For managed nodes that run RHEL 8 or earlier:

```
# cat /etc/ssh/sshd_config
...
# 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>
```

- For managed nodes that run RHEL 9 or later:

```
# cat /etc/ssh/sshd_config.d/42-my-application.conf
# Ansible managed
#
AcceptEnv LANG LS_COLORS EDITOR
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.sshd/README.md** file
- **/usr/share/doc/rhel-system-roles/sshd/** directory
- **sshd_config(5)** man page on your system

25.4. OVERRIDING THE SYSTEM-WIDE CRYPTOGRAPHIC POLICY ON AN SSH SERVER BY USING THE SSHD RHEL SYSTEM ROLE

When the default cryptographic settings do not meet certain security or compatibility needs, you may want to override the system-wide cryptographic policy on the OpenSSH server using the **sshd** RHEL system role. Especially, in the following notable situations:

- Compatibility with older clients: necessity to use weaker-than-default encryption algorithms, key exchange protocols, or ciphers.
- Enforcing stronger security policies: simultaneously, you can disable weaker algorithms. Such a measure could exceed the default system cryptographic policies, especially in the highly secure and regulated environments.
- Performance considerations: the system defaults could enforce stronger algorithms that can be computationally intensive for some systems.
- Customizing for specific security needs: adapting for unique requirements that are not covered by the default cryptographic policies.

WARNING

It is not possible to override all aspects of the cryptographic policies from the **sshd** RHEL system role. For example, SHA1 signatures might be forbidden on a different layer so for a more generic solution, see Setting a custom cryptographic policy by using RHEL system roles.

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.

Procedure

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

```
- name: Deploy SSH configuration for OpenSSH server
  hosts: managed-node-01.example.com
  tasks:
    - name: Overriding the system-wide cryptographic policy
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.sshd
      vars:
        sshd_sysconfig: true
        sshd_sysconfig_override_crypto_policy: true
        sshd_KexAlgorithms: ecdh-sha2-nistp521
        sshd_Ciphers: aes256-ctr
        sshd_MACs: hmac-sha2-512-etm@openssh.com
        sshd_HostKeyAlgorithms: rsa-sha2-512,rsa-sha2-256
```

The settings specified in the example playbook include the following:

sshd_KexAlgorithms
: You can choose key exchange algorithms, for example, **ecdh-sha2-nistp256**, **ecdh-sha2-nistp384**, **ecdh-sha2-nistp521**,**diffie-hellman-group14-sha1**, or **diffie-hellman-group-exchange-sha256**.

sshd_Ciphers
: You can choose ciphers, for example, **aes128-ctr**, **aes192-ctr**, or **aes256-ctr**.

sshd_MACs
: You can choose MACs, for example, **hmac-sha2-256**, **hmac-sha2-512**, or **hmac-sha1**.

sshd_HostKeyAlgorithms
: You can choose a public key algorithm, for example, **ecdsa-sha2-nistp256**, **ecdsa-sha2-nistp384**, **ecdsa-sha2-nistp521**, or **ssh-rsa**.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.sshd/README.md** file on the control node.

On RHEL 9 managed nodes, the system role writes the configuration into the **/etc/ssh/sshd_config.d/00-ansible_system_role.conf** file, where cryptographic options are applied automatically. You can change the file by using the **sshd_config_file** variable. However, to ensure the configuration is effective, use a file name that lexicographically precedes the **/etc/ssh/sshd_config.d/50-redhat.conf** file, which includes the configured crypto policies.

On RHEL 8 managed nodes, you must enable override by setting the **sshd_sysconfig_override_crypto_policy** and **sshd_sysconfig** variables to **true**.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- You can verify the success of the procedure by using the verbose SSH connection and check the defined variables in the following output:

```
$ ssh -vvv <ssh_server>
...
debug2: peer server KEXINIT proposal
debug2: KEX algorithms: ecdh-sha2-nistp521
debug2: host key algorithms: rsa-sha2-512,rsa-sha2-256
debug2: ciphers ctos: aes256-ctr
debug2: ciphers stoc: aes256-ctr
debug2: MACs ctos: hmac-sha2-512-etm@openssh.com
debug2: MACs stoc: hmac-sha2-512-etm@openssh.com
...
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.sshd/README.md** file
- **/usr/share/doc/rhel-system-roles/sshd/** directory

25.5. HOW THE SSH RHEL SYSTEM ROLE MAPS SETTINGS FROM A PLAYBOOK TO THE CONFIGURATION FILE

In the **ssh** RHEL system role playbook, you can define the parameters for the client SSH configuration file.

If you do not specify these settings, the role produces a global **ssh_config** file that matches the RHEL defaults.

In all the cases, booleans correctly render as **yes** or **no** in the final configuration on your managed nodes. You can use lists to define multi-line configuration items. 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.

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.ssh/README.md** file
- **/usr/share/doc/rhel-system-roles/ssh/** directory

25.6. CONFIGURING OPENSSH CLIENTS BY USING THE SSH RHEL SYSTEM ROLE

You can use the **ssh** RHEL system role to configure multiple OpenSSH clients. These enable the local user to establish a secure connection with the remote OpenSSH server by ensuring namely:

- Secure connection initiation
- Credentials provision
- Negotiation with the OpenSSH server on the encryption method used for the secure communication channel
- Ability to send files securely to and from the OpenSSH server

> **NOTE**
>
> You can use the **ssh** RHEL system role alongside with other system roles that change SSH configuration, for example the Identity Management RHEL system roles. To prevent the configuration from being overwritten, make sure that the **ssh** RHEL system role uses a drop-in directory (default in RHEL 8 and later).

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.

Procedure

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

```
---
- name: SSH client configuration
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure ssh clients
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.ssh
      vars:
        ssh_user: root
        ssh:
          Compression: true
          GSSAPIAuthentication: no
          ControlMaster: auto
          ControlPath: ~/.ssh/.cm%C
          Host:
```

```
        - Condition: example
          Hostname: server.example.com
          User: user1
      ssh_ForwardX11: no
```

The settings specified in the example playbook include the following:

ssh_user: root
: Configures the **root** user's SSH client preferences on the managed nodes with certain configuration specifics.

Compression: true
: Compression is enabled.

ControlMaster: auto
: ControlMaster multiplexing is set to **auto**.

Host
: Creates alias **example** for connecting to the **server.example.com** host as a user called **user1**.

ssh_ForwardX11: no
: X11 forwarding is disabled.

For details about the role variables and the OpenSSH configuration options used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.ssh/README.md** file and the **ssh_config(5)** manual page on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Verify that the managed node has the correct configuration by displaying the SSH configuration file:

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

Additional resources

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.ssh/README.md** file
- **/usr/share/doc/rhel-system-roles/ssh/** directory
- **ssh_config(5)** manual page

CHAPTER 26. MANAGING LOCAL STORAGE BY USING RHEL SYSTEM ROLES

To manage LVM and local file systems (FS) by 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.

26.1. CREATING AN XFS FILE SYSTEM ON A BLOCK DEVICE BY USING THE STORAGE RHEL SYSTEM ROLE

The example Ansible playbook uses the storage role to create an XFS file system on a block device using the default parameters. If the file system on the **/dev/sdb** device or the mount point directory does not exist, the playbook creates them.

> **NOTE**
>
> 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.

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.

Procedure

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

```
---
- name: Manage local storage
  hosts: managed-node-01.example.com
  tasks:
    - name: Create an XFS file system on a block device
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.storage
      vars:
        storage_volumes:
          - name: barefs
            type: disk
            disks:
              - sdb
            fs_type: xfs
```

The setting specified in the example playbook include the following:

name: barefs

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.

fs_type: *<file_system>*

You can omit the **fs_type** parameter if you want to use the default file system XFS.

disks: *<list_of_disks_and_volumes>*

A YAML list of disk and LV names. To create the file system on an LV, provide the LVM setup under the **disks** attribute, including the enclosing volume group. For details, see Creating or resizing a logical volume by using the storage RHEL system role .
Do not provide the path to the LV device.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** directory

26.2. PERSISTENTLY MOUNTING A FILE SYSTEM BY USING THE STORAGE RHEL SYSTEM ROLE

The example Ansible playbook uses the storage role to persistently mount an existing file system. It ensures that the file system is immediately available and persistently mounted by adding the appropriate entry to the **/etc/fstab** file. This allows the file system to remain mounted across reboots. If the file system on the **/dev/sdb** device or the mount point directory does not exist, the playbook creates them.

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.

Procedure

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

```
---
- name: Manage local storage
  hosts: managed-node-01.example.com
```

```
tasks:
  - name: Persistently mount a file system
    ansible.builtin.include_role:
      name: redhat.rhel_system_roles.storage
    vars:
      storage_volumes:
        - name: barefs
          type: disk
          disks:
            - sdb
          fs_type: xfs
          mount_point: /mnt/data
          mount_user: somebody
          mount_group: somegroup
          mount_mode: 0755
```

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** directory

26.3. CREATING OR RESIZING A LOGICAL VOLUME BY USING THE STORAGE RHEL SYSTEM ROLE

Use the **storage** role to perform the following tasks:

- To create an LVM logical volume in a volume group consisting of many disks
- To resize an existing file system on LVM
- To express an LVM volume size in percentage of the pool’s total size

If the volume group does not exist, the role creates it. If a logical volume exists in the volume group, it is resized if the size does not match what is specified in the playbook.

If you are reducing a logical volume, to prevent data loss you must ensure that the file system on that logical volume is not using the space in the logical volume that is being reduced.

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.

Procedure

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

```
---
- name: Manage local storage
  hosts: managed-node-01.example.com
  tasks:
    - name: Create logical volume
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.storage
      vars:
        storage_pools:
          - name: myvg
            disks:
              - sda
              - sdb
              - sdc
            volumes:
              - name: mylv
                size: 2G
                fs_type: ext4
                mount_point: /mnt/data
```

   The settings specified in the example playbook include the following:

   size: <*size*>

   You must specify the size by using units (for example, GiB) or percentage (for example, 60%).

   For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Verify that specified volume has been created or resized to the requested size:

```
# ansible managed-node-01.example.com -m command -a 'lvs myvg'
```

▪

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** directory

26.4. ENABLING ONLINE BLOCK DISCARD BY USING THE STORAGE RHEL SYSTEM ROLE

You can mount an XFS file system with the online block discard option to automatically discard unused blocks.

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.

Procedure

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

```
---
- name: Manage local storage
  hosts: managed-node-01.example.com
  tasks:
    - name: Enable online block discard
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.storage
      vars:
        storage_volumes:
          - name: barefs
            type: disk
            disks:
              - sdb
            fs_type: xfs
            mount_point: /mnt/data
            mount_options: discard
```

   For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Verify that online block discard option is enabled:

```
# ansible managed-node-01.example.com -m command -a 'findmnt /mnt/data'
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** directory

26.5. CREATING AND MOUNTING A FILE SYSTEM BY USING THE STORAGE RHEL SYSTEM ROLE

The example Ansible playbook uses the storage role to create and mount a file system. It ensures that the file system is immediately available and persistently mounted by adding the appropriate entry to the **/etc/fstab** file. This allows the file system to remain mounted across reboots. If the file system on the **/dev/sdb** device or the mount point directory does not exist, the playbook creates them.

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.

Procedure

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

```
---
- name: Manage local storage
  hosts: managed-node-01.example.com
  tasks:
    -name: Create and mount a file system
    ansible.builtin.include_role:
        name: redhat.rhel_system_roles.storage
    vars:
      storage_volumes:
        - name: barefs
          type: disk
          disks:
            - sdb
          fs_type: ext4
          fs_label: label-name
          mount_point: /mnt/data
```

The setting specified in the example playbook include the following:

disks: *<list_of_devices>*

A YAML list of device names that the role uses when it creates the volume.

fs_type: *<file_system>*

Specifies the file system the role should set on the volume. You can select **xfs**, **ext3**, **ext4**, **swap**, or **unformatted**.

label-name: *<file_system_label>*

Optional: sets the label of the file system.

mount_point: *<directory>*

Optional: if the volume should be automatically mounted, set the **mount_point** variable to the directory to which the volume should be mounted.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** directory

26.6. CONFIGURING A RAID VOLUME BY USING THESTORAGE RHEL SYSTEM ROLE

With the **storage** system role, you can configure a RAID volume on RHEL by using Red Hat Ansible Automation Platform and Ansible-Core. Create an Ansible playbook with the parameters to configure a RAID volume to suit your requirements.

> **WARNING**
>
> Device names might change in certain circumstances, for example, when you add a new disk to a system. Therefore, to prevent data loss, use persistent naming attributes in the playbook. For more information about persistent naming attributes, see Overview of persistent naming attributes.

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.

Procedure

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

```
---
- name: Manage local storage
  hosts: managed-node-01.example.com
  tasks:
    - name: Create a RAID on sdd, sde, sdf, and sdg
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.storage
      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
```

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Verify that the array was correctly created:

```
# ansible managed-node-01.example.com -m command -a 'mdadm --detail
/dev/md/data'
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** directory

- Managing RAID

26.7. CONFIGURING AN LVM POOL WITH RAID BY USING THE STORAGE RHEL SYSTEM ROLE

With the **storage** system role, you can configure an LVM pool with RAID on RHEL by using Red Hat Ansible Automation Platform. You can set up an Ansible playbook with the available parameters to configure an LVM pool with RAID.

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.

Procedure

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

```
---
- name: Manage local storage
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure LVM pool with RAID
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.storage
      vars:
        storage_safe_mode: false
        storage_pools:
          - name: my_pool
            type: lvm
            disks: [sdh, sdi]
            raid_level: raid1
            volumes:
              - name: my_volume
                size: "1 GiB"
                mount_point: "/mnt/app/shared"
                fs_type: xfs
                state: present
```

   For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Verify that your pool is on RAID:

```
# ansible managed-node-01.example.com -m command -a 'lsblk'
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** directory
- Managing RAID

26.8. CONFIGURING A STRIPE SIZE FOR RAID LVM VOLUMES BY USING THE STORAGE RHEL SYSTEM ROLE

With the **storage** system role, you can configure a stripe size for RAID LVM volumes on RHEL by using Red Hat Ansible Automation Platform. You can set up an Ansible playbook with the available parameters to configure an LVM pool with RAID.

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.

Procedure

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

```
---
- name: Manage local storage
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure stripe size for RAID LVM volumes
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.storage
      vars:
        storage_safe_mode: false
        storage_pools:
          - name: my_pool
            type: lvm
            disks: [sdh, sdi]
            volumes:
              - name: my_volume
                size: "1 GiB"
                mount_point: "/mnt/app/shared"
                fs_type: xfs
```

```
            raid_level: raid0
            raid_stripe_size: "256 KiB"
            state: present
```

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Verify that stripe size is set to the required size:

```
# ansible managed-node-01.example.com -m command -a 'lvs -o+stripesize /dev/my_pool/my_volume'
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** directory
- Managing RAID

26.9. CONFIGURING AN LVM-VDO VOLUME BY USING THE STORAGE RHEL SYSTEM ROLE

You can use the **storage** RHEL system role to create a VDO volume on LVM (LVM-VDO) with enabled compression and deduplication.

> **NOTE**
>
> Because of the **storage** system role use of LVM-VDO, only one volume can be created per pool.

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.

Procedure

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

```
---
- name: Manage local storage
  hosts: managed-node-01.example.com
  tasks:
    - name: Create LVM-VDO volume under volume group 'myvg'
      ansible.builtin.include_role:
        name: redhat.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
```

   The settings specified in the example playbook include the following:

   vdo_pool_size: *<size>*

   The actual size that the volume takes on the device. You can specify the size in human-readable format, such as 10 GiB. If you do not specify a unit, it defaults to bytes.

   size: *<size>*

   The virtual size of VDO volume.

   For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- View the current status of compression and deduplication:

```
$ ansible managed-node-01.example.com -m command -a 'lvs -
o+vdo_compression,vdo_compression_state,vdo_deduplication,vdo_index_state'
  LV       VG      Attr       LSize   Pool   Origin Data%  Meta%  Move Log Cpy%Sync Convert
VDOCompression VDOCompressionState VDODeduplication VDOIndexState
  mylv1   myvg   vwi-a-v---   3.00t vpool0                                                          enabled
online          enabled        online
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** directory

26.10. CREATING A LUKS2 ENCRYPTED VOLUME BY 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

- 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.

Procedure

1. Store your sensitive variables in an encrypted file:
    a. Create the vault:

    ```
    $ ansible-vault create ~/vault.yml
    New Vault password: <vault_password>
    Confirm New Vault password: <vault_password>
    ```

    b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

    ```
    luks_password: <password>
    ```

    c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

```
---
- name: Manage local storage
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Create and configure a volume encrypted with LUKS
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.storage
      vars:
        storage_volumes:
          - name: barefs
            type: disk
            disks:
              - sdb
```

```
          fs_type: xfs
          fs_label: <label>
          mount_point: /mnt/data
          encryption: true
          encryption_password: "{{ luks_password }}"
```

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --ask-vault-pass --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Verification

1. Find the **luksUUID** value of the LUKS encrypted volume:

```
# ansible managed-node-01.example.com -m command -a 'cryptsetup luksUUID /dev/sdb'

4e4e7970-1822-470e-b55a-e91efe5d0f5c
```

2. View the encryption status of the volume:

```
# ansible managed-node-01.example.com -m command -a 'cryptsetup status luks-4e4e7970-1822-470e-b55a-e91efe5d0f5c'

/dev/mapper/luks-4e4e7970-1822-470e-b55a-e91efe5d0f5c is active and is in use.
  type:    LUKS2
  cipher:  aes-xts-plain64
  keysize: 512 bits
  key location: keyring
  device:  /dev/sdb
...
```

3. Verify the created LUKS encrypted volume:

```
# ansible managed-node-01.example.com -m command -a 'cryptsetup luksDump /dev/sdb'

LUKS header information
Version:        2
Epoch:          3
Metadata area:  16384 [bytes]
Keyslots area:  16744448 [bytes]
UUID:           4e4e7970-1822-470e-b55a-e91efe5d0f5c
Label:          (no label)
```

```
Subsystem:      (no subsystem)
Flags:          (no flags)

Data segments:
  0: crypt
        offset: 16777216 [bytes]
        length: (whole device)
        cipher: aes-xts-plain64
        sector: 512 [bytes]
...
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** directory
- Encrypting block devices by using LUKS
- Ansible vault

26.11. CREATING SHARED LVM DEVICES USING THE STORAGE RHEL SYSTEM ROLE

You can use the **storage** RHEL system role to create shared LVM devices if you want your multiple systems to access the same storage at the same time.

This can bring the following notable benefits:

- Resource sharing
- Flexibility in managing storage resources
- Simplification of storage management tasks

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.
- **lvmlockd** is configured on the managed node. For more information, see Configuring LVM to share SAN disks among multiple machines.

Procedure

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

```
---
- name: Manage local storage
  hosts: managed-node-01.example.com
  become: true
  tasks:
```

```
    - name: Create shared LVM device
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.storage
      vars:
        storage_pools:
          - name: vg1
            disks: /dev/vdb
            type: lvm
            shared: true
            state: present
            volumes:
              - name: lv1
                size: 4g
                mount_point: /opt/test1
        storage_safe_mode: false
        storage_use_partitions: true
```

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** directory

CHAPTER 27. MANAGING SYSTEMD UNITS BY USING RHEL SYSTEM ROLES

By using the **systemd** RHEL system role, you can automate certain systemd-related tasks and perform them remotely. You can use the role for the following actions:

- Manage services
- Deploy units
- Deploy drop-in files

27.1. MANAGING SERVICES BY USING THE SYSTEMD RHEL SYSTEM ROLE

You can automate and remotely manage systemd units, such as starting or enabling services, by using the **systemd** RHEL system role.

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.

Procedure

1. Create a playbook file, for example ~/**playbook.yml**, with the following content. Use only the variables depending on what actions you want to perform.

```
---
- name: Managing systemd services
  hosts: managed-node-01.example.com
  tasks:
    - name: Perform action on systemd units
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.systemd
      vars:
        systemd_started_units:
          - <systemd_unit_1>.service
        systemd_stopped_units:
          - <systemd_unit_2>.service
        systemd_restarted_units:
          - <systemd_unit_3>.service
        systemd_reloaded_units:
          - <systemd_unit_4>.service
        systemd_enabled_units:
          - <systemd_unit_5>.service
        systemd_disabled_units:
          - <systemd_unit_6>.service
        systemd_masked_units:
```

```
  - <systemd_unit_7>.service
systemd_unmasked_units:
  - <systemd_unit_8>.service
```

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.systemd/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.systemd/README.md** file
- **/usr/share/doc/rhel-system-roles/systemd/** directory

27.2. DEPLOYING SYSTEMD DROP-IN FILES BY USING THE SYSTEMD RHEL SYSTEM ROLE

Systemd applies drop-in files on top of setting it reads for a unit from other locations. Therefore, you can modify unit settings with drop-in files without changing the original unit file. By using the **systemd** RHEL system role, you can automate the process of deploying drop-in files.

IMPORTANT

The role uses the hard-coded file name **99-override.conf** to store drop-in files in **/etc/systemd/system/*<name>*._<unit_type>/**. Note that it overrides existing files with this name in the destination directory.

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.

Procedure

1. Create a Jinja2 template with the systemd drop-in file contents. For example, create the **~/sshd.service.conf.j2** file with the following content:

```
{{ ansible_managed | comment }}
[Unit]
After=
After=network.target sshd-keygen.target network-online.target
```

This drop-in file specifies the same units in the **After** setting as the original **/usr/lib/systemd/system/sshd.service** file and, additionally, **network-online.target**. With this extra target, **sshd** starts after the network interfaces are actived and have IP addresses assigned. This ensures that **sshd** can bind to all IP addresses.

Use the ***<name>.<unit_type>*.conf.j2** convention for the file name. For example, to add a drop-in for the **sshd.service** unit, you must name the file **sshd.service.conf.j2**. Place the file in the same directory as the playbook.

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

```
---
- name: Managing systemd services
  hosts: managed-node-01.example.com
  tasks:
    - name: Deploy an sshd.service systemd drop-in file
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.systemd
      vars:
        systemd_dropins:
          - sshd.service.conf.j2
```

The settings specified in the example playbook include the following:

systemd_dropins: *<list_of_files>*
: Specifies the names of the drop-in files to deploy in YAML list format.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.systemd/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Verify that the role placed the drop-in file in the correct location:

```
# ansible managed-node-01.example.com -m command -a 'ls
/etc/systemd/system/sshd.service.d/'
99-override.conf
```

Additional resources

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

- **/usr/share/doc/rhel-system-roles/systemd/** directory

27.3. DEPLOYING SYSTEMD SYSTEM UNITS BY USING THE SYSTEMD RHEL SYSTEM ROLE

You can create unit files for custom applications, and systemd reads them from the **/etc/systemd/system/** directory. By using the **systemd** RHEL system role, you can automate the deployment of custom unit files.

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.

Procedure

1. Create a Jinja2 template with the custom systemd unit file contents. For example, create the **~/example.service.j2** file with the contents for your service:

```
{{ ansible_managed | comment }}
[Unit]
Description=Example systemd service unit file

[Service]
ExecStart=/bin/true
```

   Use the ***<name>.<unit_type>*.j2** convention for the file name. For example, to create the **example.service** unit, you must name the file **example.service.j2**. Place the file in the same directory as the playbook.

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

```
---
- name: Managing systemd services
  hosts: managed-node-01.example.com
  tasks:
    - name: Deploy, enable, and start a custom systemd service
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.systemd
      vars:
        systemd_unit_file_templates:
          - example.service.j2
        systemd_enabled_units:
          - example.service
        systemd_started_units:
          - example.service
```

   For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.systemd/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Verify that the service is enabled and started:

```
# ansible managed-node-01.example.com -m command -a 'systemctl status example.service'
...
● example.service - A service for demonstrating purposes
  Loaded: loaded (/etc/systemd/system/example.service; enabled; vendor preset: disabled)
  Active: active (running) since Thu 2024-07-04 15:59:18 CEST; 10min ago
...
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.systemd/README.md** file
- **/usr/share/doc/rhel-system-roles/systemd/** directory

CHAPTER 28. CONFIGURING TIME SYNCHRONIZATION BY USING RHEL SYSTEM ROLES

The Network Time Protocol (NTP) and Precision Time Protocol (PTP) are standards to synchronize the clock of computers over a network. An accurate time synchronization in networks is important because certain services rely on it. For example, Kerberos tolerates only a small time difference between the server and client to prevent replay attacks.

You can set the time service to configure in the **timesync_ntp_provider** variable of a playbook. If you do not set this variable, the role determines the time service based on the following factors:

- On RHEL 8 and later: **chronyd**
- On RHEL 6 and 7: **chronyd** (default) or, if already installed **ntpd**.

28.1. CONFIGURING TIME SYNCHRONIZATION OVER NTP BY USING THE TIMESYNC RHEL SYSTEM ROLE

The Network Time Protocol (NTP) synchronizes the time of a host with an NTP server over a network. In IT networks, services rely on a correct system time, for example, for security and logging purposes. By using the **timesync** RHEL system role, you can automate the configuration of Red Hat Enterprise Linux NTP clients in your network and keep the time synchronized.

> **WARNING**
>
> The **timesync** RHEL system role replaces the configuration of the specified given or detected provider service on the managed host. Consequently, all settings are lost if they are not specified in the playbook.

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.

Procedure

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

```
---
- name: Managing time synchronization
  hosts: managed-node-01.example.com
  tasks:
    - name: Configuring NTP with an internal server (preferred) and a public server pool as
fallback
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.timesync
```

```
vars:
  timesync_ntp_servers:
  - hostname: time.example.com
    trusted: yes
    prefer: yes
    iburst: yes
  - hostname: 0.rhel.pool.ntp.org
    pool: yes
    iburst: yes
```

The settings specified in the example playbook include the following:

pool: *<yes/no>*
: Flags a source as an NTP pool rather than an individual host. In this case, the service expects that the name resolves to multiple IP addresses which can change over time.

iburst: *yes*
: Enables fast initial synchronization.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.timesync/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Display the details about the time sources:
  - If the managed node runs the **chronyd** service, enter:

```
# ansible managed-node-01.example.com -m command -a 'chronyc sources'
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
=======
^* time.example.com             1  10   377   210   +159us[  +55us] +/-   12ms
^? ntp.example.org             2   9   377   409  +1120us[+1021us] +/-   42ms
^? time.us.example.net          2   9   377   992   -329us[ -386us] +/-   15ms
...
```

  - If the managed node runs the **ntpd** service, enter:

```
# ansible managed-node-01.example.com -m command -a 'ntpq -p'
     remote           refid      st t when poll reach   delay   offset  jitter
===============================================================================
======
*time.example.com .PTB.           1 u    2   64   77   23.585  967.902   0.684
```

```
- ntp.example.or 192.0.2.17       2 u   -   64   77   27.090  966.755   0.468
+time.us.example 198.51.100.19    2 u   65   64   37   18.497  968.463   1.588
...
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.time_sync/README.md** file
- **/usr/share/doc/rhel-system-roles/time_sync/** directory
- Are the rhel.pool.ntp.org NTP servers supported by Red Hat? (Red Hat Knowledgebase)

28.2. CONFIGURING TIME SYNCHRONIZATION OVER NTP WITH NTS BY USING THE TIMESYNC RHEL SYSTEM ROLE

The Network Time Protocol (NTP) synchronizes the time of a host with an NTP server over a network. By using the Network Time Security (NTS) mechanism, clients establish a TLS-encrypted connection to the server and authenticate NTP packets. In IT networks, services rely on a correct system time, for example, for security and logging purposes. By using the **timesync** RHEL system role, you can automate the configuration of Red Hat Enterprise Linux NTP clients in your network and keep the time synchronized over NTS.

Note that you cannot mix NTS servers with non-NTS servers. In mixed configurations, NTS servers are trusted and clients do not fall back to unauthenticated NTP sources because they can be exploited in man-in-the-middle (MITM) attacks. For further details, see the **authselectmode** parameter description in the **chrony.conf(5)** man page on your system.

> **WARNING**
>
> The **timesync** RHEL system role replaces the configuration of the specified given or detected provider service on the managed host. Consequently, all settings are lost if they are not specified in the playbook.

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 managed nodes use **chronyd**.

Procedure

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

```
---
- name: Managing time synchronization
  hosts: managed-node-01.example.com
```

```
tasks:
  - name: Configuring NTP with NTS-enabled servers
    ansible.builtin.include_role:
      name: redhat.rhel_system_roles.timesync
    vars:
      timesync_ntp_servers:
      - hostname: ptbtime1.ptb.de
        nts: yes
        iburst: yes
```

The settings specified in the example playbook include the following:

iburst: *yes*

Enables fast initial synchronization.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.timesync/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- If the managed node runs the **chronyd** service:

1. Display the details about the time sources:

```
# ansible managed-node-01.example.com -m command -a 'chronyc sources'
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
=======
^* ptbtime1.ptb.de              1   6    17    55    -13us[  -54us] +/-   12ms
^- ptbtime2.ptb.de              1   6    17    56  -257us[ -297us] +/-   12ms
```

2. For sources with NTS enabled, display information that is specific to authentication of NTP sources:

```
# ansible managed-node-01.example.com -m command -a 'chronyc -N authdata'
Name/IP address             Mode KeyID Type KLen Last Atmp  NAK Cook CLen
===============================================================================
=
ptbtime1.ptb.de              NTS     1   15  256  229    0    0    8  100
ptbtime2.ptb.de              NTS     1   15  256  230    0    0    8  100
```

Verify that the reported cookies in the **Cook** column is larger than 0.

- If the managed node runs the **ntpd** service, enter:

```
# ansible managed-node-01.example.com -m command -a 'ntpq -p'
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
===
*ptbtime1.ptb.de .PTB.           1 8    2   64   77   23.585  967.902   0.684
-ptbtime2.ptb.de .PTB.           1 8   30   64   78   24.653  993.937   0.765
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.time_sync/README.md** file
- **/usr/share/doc/rhel-system-roles/time_sync/** directory
- Are the rhel.pool.ntp.org NTP servers supported by Red Hat? (Red Hat Knowledgebase)

CHAPTER 29. CONFIGURING A SYSTEM FOR SESSION RECORDING BY USING RHEL SYSTEM ROLES

Use the **tlog** RHEL system role to record and monitor terminal session activities on your managed nodes in an automatic fashion. You can configure the recording to take place per user or user group by means of the **SSSD** service.

The session recording solution in the tlog RHEL system role consists of the following components:

- The **tlog** utility
- System Security Services Daemon (SSSD)
- Optional: The web console interface

29.1. CONFIGURING SESSION RECORDING FOR INDIVIDUAL USERS BY USING THE TLOG RHEL SYSTEM ROLE

Prepare and apply an Ansible playbook to configure a RHEL system to log session recording data to the **systemd** journal.

With that, you can enable recording the terminal output and input of a specific user during their sessions, when the user logs in on the console, or by SSH.

The playbook installs **tlog-rec-session**, a terminal session I/O logging program, that acts as the login shell for a user. The role creates an SSSD configuration drop file, and this file defines for which users and groups the login shell should be used. 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.

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.

Procedure

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

```
---
- name: Deploy session recording
  hosts: managed-node-01.example.com
  tasks:
    - name: Enable session recording for specific users
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.tlog
  vars:
    tlog_scope_sssd: some
    tlog_users_sssd:
      - <recorded_user>
```

tlog_scope_sssd: *<value>*
: The **some** value specifies you want to record only certain users and groups, not **all** or **none**.

tlog_users_sssd: *<list_of_users>*
: A YAML list of users you want to record a session from. Note that the role does not add users if they do not exist.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

1. Check the SSSD drop-in file's content:

```
# cd /etc/sssd/conf.d/sssd-session-recording.conf
```

You can see that the file contains the parameters you set in the playbook.

2. Log in as a user whose session will be recorded, perform some actions, and log out.

3. As the **root** user:

a. Display the list of recorded sessions:

```
# journalctl _COMM=tlog-rec-sessio
Nov 12 09:17:30 managed-node-01.example.com -tlog-rec-session[1546]:
{"ver":"2.3","host":"managed-node-
01.example.com","rec":"07418f2b0f334c1696c10cbe6f6f31a6-60a-e4a2","user":"demo-
user",...
...
```

You require the value of the **rec** (recording ID) field in the next step.

Note that the value of the **_COMM** field is shortened due to a 15 character limit.

b. Play back a session:

```
# tlog-play -r journal -M TLOG_REC=<recording_id>
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.tlog/README.md** file
- **/usr/share/doc/rhel-system-roles/tlog/** directory

29.2. EXCLUDING CERTAIN USERS AND GROUPS FROM SESSION RECORDING BY USING THE TLOG RHEL SYSTEM ROLE

You can use the **tlog_exclude_users_sssd** and **tlog_exclude_groups_sssd** role variables from the **tlog** RHEL system role to exclude users or groups from having their sessions recorded and logged in the **systemd** journal.

The playbook installs **tlog-rec-session**, a terminal session I/O logging program, that acts as the login shell for a user. The role creates an SSSD configuration drop file, and this file defines for which users and groups the login shell should be used. 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.

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.

Procedure

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

```
---
- name: Deploy session recording excluding users and groups
  hosts: managed-node-01.example.com
  tasks:
    - name: Exclude users and groups
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.tlog
      vars:
        tlog_scope_sssd: all
        tlog_exclude_users_sssd:
          - jeff
          - james
        tlog_exclude_groups_sssd:
          - admins
```

tlog_scope_sssd: *<value>*

The value **all** specifies that you want to record all users and groups.

tlog_exclude_users_sssd: *<user_list>*

A YAML list of users user names you want to exclude from the session recording.

tlog_exclude_groups_sssd: *<group_list>*

A YAML list of groups you want to exclude from the session recording.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

1. Check the SSSD drop-in file's content:

```
# cat /etc/sssd/conf.d/sssd-session-recording.conf
```

You can see that the file contains the parameters you set in the playbook.

2. Log in as a user whose session will be recorded, perform some actions, and log out.

3. As the **root** user:

   a. Display the list of recorded sessions:

```
# journalctl _COMM=tlog-rec-sessio
Nov 12 09:17:30 managed-node-01.example.com -tlog-rec-session[1546]:
{"ver":"2.3","host":"managed-node-
01.example.com","rec":"07418f2b0f334c1696c10cbe6f6f31a6-60a-e4a2","user":"demo-
user",...
...
```

   You require the value of the **rec** (recording ID) field in the next step.

   Note that the value of the **_COMM** field is shortened due to a 15 character limit.

   b. Play back a session:

```
# tlog-play -r journal -M TLOG_REC=<recording_id>
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.tlog/README.md** file
- **/usr/share/doc/rhel-system-roles/tlog/** directory

CHAPTER 30. CONFIGURING VPN CONNECTIONS BY USING RHEL SYSTEM ROLES

A VPN is an encrypted connection to securely transmit traffic over untrusted networks. By using the **vpn** RHEL system role, you can automate the process of creating VPN configurations.

> **NOTE**
>
> The **vpn** RHEL system role supports only Libreswan, which is an IPsec implementation, as the VPN provider.

30.1. CREATING A HOST-TO-HOST IPSEC VPN WITH PSK AUTHENTICATION BY USING THE VPN RHEL SYSTEM ROLE

You can use IPsec to directly connect hosts to each other through a VPN. The hosts can use a pre-shared key (PSK) to authenticate to each other. By using the **vpn** RHEL system role, you can automate the process of creating IPsec host-to-host connections with PSK authentication.

By default, the role creates a tunnel-based VPN.

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.

Procedure

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

```
---
- name: Configuring VPN
  hosts: managed-node-01.example.com, managed-node-02.example.com
  tasks:
    - name: IPsec VPN with PSK authentication
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.vpn
      vars:
        vpn_connections:
          - hosts:
              managed-node-01.example.com:
              managed-node-02.example.com:
            auth_method: psk
            auto: start
        vpn_manage_firewall: true
        vpn_manage_selinux: true
```

The settings specified in the example playbook include the following:

hosts: *<list>*

Defines a YAML dictionary with the hosts between which you want to configure a VPN. If an entry is not an Ansible managed node, you must specify its fully-qualified domain name (FQDN) or IP address in the **hostname** parameter, for example:

```
...
- hosts:
    ...
    external-host.example.com:
      hostname: 192.0.2.1
```

The role configures the VPN connection on each managed node. The connections are named ***<host_A>*-to-*<host_B>***, for example, **managed-node-01.example.com-to-managed-node-02.example.com**. Note that the role can not configure Libreswan on external (unmanaged) nodes. You must manually create the configuration on these hosts.

auth_method: psk

Enables PSK authentication between the hosts. The role uses **openssl** on the control node to create the PSK.

auto: *<start-up_method>*

Specifies the start-up method of the connection. Valid values are **add**, **ondemand**, **start**, and **ignore**. For details, see the **ipsec.conf(5)** man page on a system with Libreswan installed. The default value of this variable is null, which means no automatic startup operation.

vpn_manage_firewall: true

Defines that the role opens the required ports in the **firewalld** service on the managed nodes.

vpn_manage_selinux: true

Defines that the role sets the required SELinux port type on the IPsec ports.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.vpn/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Confirm that the connections are successfully started, for example:

```
# ansible managed-node-01.example.com -m shell -a 'ipsec trafficstatus | grep "managed-node-01.example.com-to-managed-node-02.example.com"'
...
006 #3: "managed-node-01.example.com-to-managed-node-02.example.com", type=ESP, add_time=1741857153, inBytes=38622, outBytes=324626, maxBytes=2^63B, id='@managed-node-02.example.com'
```

Note that this command only succeeds if the VPN connection is active. If you set the **auto** variable in the playbook to a value other than **start**, you might need to manually activate the connection on the managed nodes first.

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.vpn/README.md** file
- **/usr/share/doc/rhel-system-roles/vpn/** directory

30.2. CREATING A HOST-TO-HOST IPSEC VPN WITH PSK AUTHENTICATION AND SEPARATE DATA AND CONTROL PLANES BY USING THE VPN RHEL SYSTEM ROLE

You can use IPsec to directly connect hosts to each other through a VPN. For example, to enhance the security by minimizing the risk of control messages being intercepted or disrupted, you can configure separate connections for both the data traffic and the control traffic. By using the **vpn** RHEL system role, you can automate the process of creating IPsec host-to-host connections with a separate data and control plane and PSK authentication.

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.

Procedure

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

```
---
- name: Configuring VPN
  hosts: managed-node-01.example.com, managed-node-02.example.com
  tasks:
    - name: IPsec VPN with PSK authentication
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.vpn
      vars:
        vpn_connections:
          - name: control_plane_vpn
            hosts:
              managed-node-01.example.com:
                hostname: 203.0.113.1  # IP address for the control plane
              managed-node-02.example.com:
                hostname: 198.51.100.2 # IP address for the control plane
            auth_method: psk
            auto: start
          - name: data_plane_vpn
            hosts:
              managed-node-01.example.com:
                hostname: 10.0.0.1   # IP address for the data plane
              managed-node-02.example.com:
```

```
        hostname: 172.16.0.2 # IP address for the data plane
      auth_method: psk
      auto: start
    vpn_manage_firewall: true
    vpn_manage_selinux: true
```

The settings specified in the example playbook include the following:

hosts: *<list>*

Defines a YAML dictionary with the hosts between which you want to configure a VPN. The connections are named ***<name>*-*<IP_address_A>*-to-*<IP_address_B>***, for example **control_plane_vpn-203.0.113.1-to-198.51.100.2**.
The role configures the VPN connection on each managed node. Note that the role can not configure Libreswan on external (unmanaged) nodes. You must manually create the configuration on these hosts.

auth_method: psk

Enables PSK authentication between the hosts. The role uses **openssl** on the control node to create the pre-shared key.

auto: *<start-up_method>*

Specifies the start-up method of the connection. Valid values are **add**, **ondemand**, **start**, and **ignore**. For details, see the **ipsec.conf(5)** man page on a system with Libreswan installed. The default value of this variable is null, which means no automatic startup operation.

vpn_manage_firewall: true

Defines that the role opens the required ports in the **firewalld** service on the managed nodes.

vpn_manage_selinux: true

Defines that the role sets the required SELinux port type on the IPsec ports.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.vpn/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Confirm that the connections are successfully started, for example:

```
# ansible managed-node-01.example.com -m shell -a 'ipsec trafficstatus | grep
"control_plane_vpn-203.0.113.1-to-198.51.100.2"'
...
006 #3: "control_plane_vpn-203.0.113.1-to-198.51.100.2", type=ESP,
add_time=1741860073, inBytes=0, outBytes=0, maxBytes=2^63B, id='198.51.100.2'
```

Note that this command only succeeds if the VPN connection is active. If you set the **auto** variable in the playbook to a value other than **start**, you might need to manually activate the connection on the managed nodes first.

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.vpn/README.md** file
- **/usr/share/doc/rhel-system-roles/vpn/** directory

30.3. CREATING AN IPSEC MESH VPN AMONG MULTIPLE HOSTS WITH CERTIFICATE-BASED AUTHENTICATION BY USING THE VPN RHEL SYSTEM ROLE

Libreswan supports creating an opportunistic mesh to establish IPsec connections among a large number of hosts with a single configuration on each host. Adding hosts to the mesh does not require updating the configuration on existing hosts. For enhanced security, use certificate-based authentication in Libreswan.

By using the **vpn** RHEL system role, you can automate configuring a VPN mesh with certificate-based authentication among managed nodes.

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.
- You prepared a PKCS #12 file for each managed node:
  - Each file contains:
    - The certificate authority (CA) certificate
    - The node’s private key
    - The node’s client certificate
  - The files are named ***<managed_node_name_as_in_the_inventory>*.p12**.
  - The files are stored in the same directory as the playbook.

Procedure

1. Edit the **~/inventory** file, and append the **cert_name** variable:

```
managed-node-01.example.com cert_name=managed-node-01.example.com
managed-node-02.example.com cert_name=managed-node-02.example.com
managed-node-03.example.com cert_name=managed-node-03.example.com
```

   Set the **cert_name** variable to the value of the common name (CN) field used in the certificate for each host. Typically, the CN field is set to the fully-qualified domain name (FQDN).

2. Store your sensitive variables in an encrypted file:

   a. Create the vault:

      ```
      $ ansible-vault create ~/vault.yml
      New Vault password: <vault_password>
      Confirm New Vault password: <vault_password>
      ```

   b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

      ```
      pkcs12_pwd: <password>
      ```

   c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

```
- name: Configuring VPN
  hosts: managed-node-01.example.com, managed-node-02.example.com, managed-node-03.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Install LibreSwan
      ansible.builtin.package:
        name: libreswan
        state: present

    - name: Identify the path to IPsec NSS database
      ansible.builtin.set_fact:
        nss_db_dir: "{{ '/etc/ipsec.d/' if
          ansible_distribution in ['CentOS', 'RedHat']
          and ansible_distribution_major_version is version('8', '=')
          else '/var/lib/ipsec/nss/' }}"

    - name: Locate IPsec NSS database files
      ansible.builtin.find:
        paths: "{{ nss_db_dir }}"
        patterns: "*.db"
      register: db_files

    - name: Remove IPsec NSS database files
      ansible.builtin.file:
        path: "{{ item.path }}"
        state: absent
      loop: "{{ db_files.files }}"
      when: db_files.matched > 0

    - name: Initialize IPsec NSS database
      ansible.builtin.command:
        cmd: ipsec initnss

    - name: Copy PKCS #12 file to the managed node
      ansible.builtin.copy:
        src: "~/{{ inventory_hostname }}.p12"
```

```
      dest: "/etc/ipsec.d/{{ inventory_hostname }}.p12"
      mode: 0600

  - name: Import PKCS #12 file in IPsec NSS database
    ansible.builtin.shell:
      cmd: 'pk12util -d {{ nss_db_dir }} -i /etc/ipsec.d/{{ inventory_hostname }}.p12 -W "{{
pkcs12_pwd }}"'

  - name: Remove PKCS #12 file
    ansible.builtin.file:
      path: "/etc/ipsec.d/{{ inventory_hostname }}.p12"
      state: absent

  - name: Opportunistic mesh IPsec VPN with certificate-based authentication
    ansible.builtin.include_role:
      name: redhat.rhel_system_roles.vpn
    vars:
      vpn_connections:
        - opportunistic: true
          auth_method: cert
          policies:
            - policy: private
              cidr: default
            - policy: private
              cidr: 192.0.2.0/24
            - policy: clear
              cidr: 192.0.2.1/32
      vpn_manage_firewall: true
      vpn_manage_selinux: true
```

The settings specified in the example playbook include the following:

opportunistic: true

Enables an opportunistic mesh among multiple hosts. The **policies** variable defines for which subnets and hosts traffic must or or can be encrypted and which of them should continue using clear text connections.

auth_method: cert

Enables certificate-based authentication. This requires that you specified the nickname of each managed node's certificate in the inventory.

policies: *<list_of_policies>*

Defines the Libreswan policies in YAML list format.
The default policy is **private-or-clear**. To change it to **private**, the above playbook contains an according policy for the default **cidr** entry.

To prevent a loss of the SSH connection during the execution of the playbook if the Ansible control node is in the same IP subnet as the managed nodes, add a **clear** policy for the control node's IP address. For example, if the mesh should be configured for the **192.0.2.0/24** subnet and the control node uses the IP address **192.0.2.1**, you require a **clear** policy for **192.0.2.1/32** as shown in the playbook.

For details about policies, see the **ipsec.conf(5)** man page on a system with Libreswan installed.

vpn_manage_firewall: true

Defines that the role opens the required ports in the **firewalld** service on the managed nodes.

vpn_manage_selinux: true

Defines that the role sets the required SELinux port type on the IPsec ports.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.vpn/README.md** file on the control node.

4. Validate the playbook syntax:

```
$ ansible-playbook --ask-vault-pass --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

5. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Verification

1. On a node in the mesh, ping another node to activate the connection:

```
[root@managed-node-01]# ping managed-node-02.example.com
```

2. Confirm that the connections is active:

```
[root@managed-node-01]# ipsec trafficstatus
006 #2: "private#192.0.2.0/24"[1] ...192.0.2.2, type=ESP, add_time=1741938929,
inBytes=372408, outBytes=545728, maxBytes=2^63B, id='CN=managed-node-
02.example.com'
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.vpn/README.md** file
- **/usr/share/doc/rhel-system-roles/vpn/** directory

CHAPTER 31. CONFIGURING MICROSOFT SQL SERVER BY USING RHEL SYSTEM ROLES

You can use the **microsoft.sql.server** Ansible system role to automate the installation and management of Microsoft SQL Server. This role also optimizes Red Hat Enterprise Linux (RHEL) to improve the performance and throughput of SQL Server by applying the **mssql** TuneD profile.

> **NOTE**
>
> During the installation, the role adds repositories for SQL Server and related packages to the managed hosts. Packages in these repositories are provided, maintained, and hosted by Microsoft.

31.1. INSTALLING AND CONFIGURING SQL SERVER WITH AN EXISTING TLS CERTIFICATE BY USING THE MICROSOFT.SQL.SERVER ANSIBLE SYSTEM ROLE

If your application requires a Microsoft SQL Server database, you can configure SQL Server with TLS encryption to enable secure communication between the application and the database. By using the **microsoft.sql.server** Ansible system role, you can automate this process and remotely install and configure SQL Server with TLS encryption. In the playbook, you can use an existing private key and a TLS certificate that was issued by a certificate authority (CA).

Depending on the RHEL version on the managed host, the version of SQL Server that you can install differs:

- RHEL 7.9: SQL Server 2017 and 2019
- RHEL 8: SQL Server 2017, 2019, and 2022
- RHEL 9.4 and later: SQL Server 2022

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.
- You installed the **ansible-collection-microsoft-sql** package or the **microsoft.sql** collection on the control node.
- The managed node has 2 GB or more RAM installed.
- The managed node uses one of the following versions: RHEL 7.9, RHEL 8, RHEL 9.4 or later.
- You stored the certificate in the **sql_crt.pem** file in the same directory as the playbook.
- You stored the private key in the **sql_cert.key** file in the same directory as the playbook.
- SQL clients trust the CA that issued the certificate.

Procedure

1. Store your sensitive variables in an encrypted file:

   a. Create the vault:

```
$ ansible-vault create ~/vault.yml
New Vault password: <vault_password>
Confirm New Vault password: <vault_password>
```

   b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

```
sa_pwd: <sa_password>
```

   c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

```
---
- name: Installing and configuring Microsoft SQL Server
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: SQL Server with an existing private key and certificate
      ansible.builtin.include_role:
        name: microsoft.sql.server
      vars:
        mssql_accept_microsoft_odbc_driver_for_sql_server_eula: true
        mssql_accept_microsoft_cli_utilities_for_sql_server_eula: true
        mssql_accept_microsoft_sql_server_standard_eula: true

        mssql_version: 2022
        mssql_password: "{{ sa_pwd }}"
        mssql_edition: Developer
        mssql_tcp_port: 1433
        mssql_manage_firewall: true

        mssql_tls_enable: true
        mssql_tls_self_sign: false
        mssql_tls_cert: sql_crt.pem
        mssql_tls_private_key: sql_cert.key
        mssql_tls_version: 1.2
        mssql_tls_force: true
```

The settings specified in the example playbook include the following:

mssql_tls_enable: true

Enables TLS encryption. If you enable this setting, you must also define **mssql_tls_cert** and **mssql_tls_private_key**.

mssql_tls_self_sign: false

Indicates whether the certificates that you use are self-signed or not. Based on this setting, the role decides whether to run the **sqlcmd** command with the **-C** argument to trust certificates.

mssql_tls_cert: *<path>*

Sets the path to the TLS certificate stored on the control node. The role copies this file to the **/etc/pki/tls/certs/** directory on the managed node.

mssql_tls_private_key: *<path>*

Sets the path to the TLS private key on the control node. The role copies this file to the **/etc/pki/tls/private/** directory on the managed node.

mssql_tls_force: true

Replaces the TLS certificate and private key in their destination directories if they exist.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/microsoft.sql-server/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --ask-vault-pass --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Verification

- On the SQL Server host, use the **sqlcmd** utility with the **-N** parameter to establish an encrypted connection to SQL server and run a query, for example:

```
$ /opt/mssql-tools/bin/sqlcmd -N -S server.example.com -U "sa" -P <sa_password> -Q
'SELECT SYSTEM_USER'
```

If the command succeeds, the connection to the server was TLS encrypted.

Additional resources

- **/usr/share/ansible/roles/microsoft.sql-server/README.md** file
- Ansible vault

31.2. INSTALLING AND CONFIGURING SQL SERVER WITH A TLS CERTIFICATE ISSUED FROM IDM BY USING THE MICROSOFT.SQL.SERVER ANSIBLE SYSTEM ROLE

If your application requires a Microsoft SQL Server database, you can configure SQL Server with TLS encryption to enable secure communication between the application and the database. If the SQL Server host is a member in a Red Hat Enterprise Linux Identity Management (IdM) domain, the **certmonger** service can manage the certificate request and future renewals.

By using the **microsoft.sql.server** Ansible system role, you can automate this process. You can remotely install and configure SQL Server with TLS encryption, and the **microsoft.sql.server** role uses the **certificate** Ansible system role to configure **certmonger** and request a certificate from IdM.

Depending on the RHEL version on the managed host, the version of SQL Server that you can install differs:

- RHEL 7.9: SQL Server 2017 and 2019
- RHEL 8: SQL Server 2017, 2019, and 2022
- RHEL 9.4 and later: SQL Server 2022

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.
- You installed the **ansible-collection-microsoft-sql** package or the **microsoft.sql** collection on the control node.
- The managed node has 2 GB or more RAM installed.
- The managed node uses one of the following versions: RHEL 7.9, RHEL 8, RHEL 9.4 or later.
- You enrolled the managed node in a Red Hat Enterprise Linux Identity Management (IdM) domain.

Procedure

1. Store your sensitive variables in an encrypted file:

   a. Create the vault:

   ```
   $ ansible-vault create ~/vault.yml
   New Vault password: <vault_password>
   Confirm New Vault password: <vault_password>
   ```

   b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

   ```
   sa_pwd: <sa_password>
   ```

   c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

```
---
- name: Installing and configuring Microsoft SQL Server
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: SQL Server with certificates issued by Red Hat IdM
      ansible.builtin.include_role:
        name: microsoft.sql.server
      vars:
```

```
mssql_accept_microsoft_odbc_driver_for_sql_server_eula: true
mssql_accept_microsoft_cli_utilities_for_sql_server_eula: true
mssql_accept_microsoft_sql_server_standard_eula: true

mssql_version: 2022
mssql_password: "{{ sa_pwd  }}"
mssql_edition: Developer
mssql_tcp_port: 1433
mssql_manage_firewall: true

mssql_tls_enable: true
mssql_tls_certificates:
  - name: sql_cert
    dns: server.example.com
    ca: ipa
```

The settings specified in the example playbook include the following:

mssql_tls_enable: true

Enables TLS encryption. If you enable this setting, you must also define **mssql_tls_certificates**.

mssql_tls_certificates

A list of YAML dictionaries with settings for the **certificate** role.

name: *<file_name>*

Defines the base name of the certificate and private key. The **certificate** role stores the certificate in the **/etc/pki/tls/certs/*<file_name>*.crt** and the private key in the **/etc/pki/tls/private/*<file_name>*.key** file.

dns: *<hostname_or_list_of_hostnames>*

Sets the hostnames that the Subject Alternative Names (SAN) field in the issued certificate contains. You can use a wildcard (*) or specify multiple names in YAML list format.

ca: *<ca_type>*

Defines how the **certificate** role requests the certificate. Set the variable to **ipa** if the host is enrolled in an IdM domain or **self-sign** to request a self-signed certificate.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/microsoft.sql-server/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --ask-vault-pass --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Verification

- On the SQL Server host, use the **sqlcmd** utility with the **-N** parameter to establish an encrypted connection to SQL server and run a query, for example:

```
$ /opt/mssql-tools/bin/sqlcmd -N -S server.example.com -U "sa" -P <sa_password> -Q 'SELECT SYSTEM_USER'
```

If the command succeeds, the connection to the server was TLS encrypted.

Additional resources

- **/usr/share/ansible/roles/microsoft.sql-server/README.md** file
- Requesting certificates by using RHEL system roles
- Ansible vault

31.3. INSTALLING AND CONFIGURING SQL SERVER WITH CUSTOM STORAGE PATHS BY USING THE MICROSOFT.SQL.SERVER ANSIBLE SYSTEM ROLE

When you use the **microsoft.sql.server** Ansible system role to install and configure a new SQL Server, you can customize the paths and modes of the data and log directories. For example, configure custom paths if you want to store databases and log files in a different directory with more storage.

IMPORTANT

If you change the data or log path and re-run the playbook, the previously-used directories and all their content remains at the original path. Only new databases and logs are stored in the new location.

Table 31.1. SQL Server default settings for data and log directories

Type	Directory	Mode	Owner	Group
Data	**/var/opt/mssql/data/**	[a]	**mssql**	**mssql**
Logs	**/var/opt/mssql/los/**	[a]	**mssql**	**mssql**

[a] If the directory exists, the role preserves the mode. If the directory does not exist, the role applies the default **umask** on the managed node when it creates the directory.

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.
- You installed the **ansible-collection-microsoft-sql** package or the **microsoft.sql** collection on the control node.
- The managed node has 2 GB or more RAM installed.

- The managed node uses one of the following versions: RHEL 7.9, RHEL 8, RHEL 9.4 or later.

Procedure

1. Store your sensitive variables in an encrypted file:

   a. Create the vault:

   ```
   $ ansible-vault create ~/vault.yml
   New Vault password: <vault_password>
   Confirm New Vault password: <vault_password>
   ```

   b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

   ```
   sa_pwd: <sa_password>
   ```

   c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

2. Edit an existing playbook file, for example ~/**playbook.yml**, and add the storage and log-related variables:

```
---
- name: Installing and configuring Microsoft SQL Server
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: SQL Server with custom storage paths
      ansible.builtin.include_role:
        name: microsoft.sql.server
      vars:
        mssql_accept_microsoft_odbc_driver_for_sql_server_eula: true
        mssql_accept_microsoft_cli_utilities_for_sql_server_eula: true
        mssql_accept_microsoft_sql_server_standard_eula: true

        mssql_version: 2022
        mssql_password: "{{ sa_pwd  }}"
        mssql_edition: Developer
        mssql_tcp_port: 1433
        mssql_manage_firewall: true

        mssql_datadir: /var/lib/mssql/
        mssql_datadir_mode: '0700'
        mssql_logdir: /var/log/mssql/
        mssql_logdir_mode: '0700'
```

The settings specified in the example playbook include the following:

mssql_datadir_mode and **mssql_logdir_mode**

Set the permission modes. Specify the value in single quotes to ensure that the role parses the value as a string and not as an octal number.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/microsoft.sql-server/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --ask-vault-pass --syntax-check ~/playbook.yml
```

   Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Verification

1. Display the mode of the data directory:

```
$ ansible managed-node-01.example.com -m command -a 'ls -ld /var/lib/mssql/'
drwx------. 12 mssql mssql 4096 Jul  3 13:53 /var/lib/mssql/
```

2. Display the mode of the log directory:

```
$ ansible managed-node-01.example.com -m command -a 'ls -ld /var/log/mssql/'
drwx------. 12 mssql mssql 4096 Jul  3 13:53 /var/log/mssql/
```

Additional resources

- **/usr/share/ansible/roles/microsoft.sql-server/README.md** file
- Ansible vault

31.4. INSTALLING AND CONFIGURING SQL SERVER WITH AD INTEGRATION BY USING THE MICROSOFT.SQL.SERVER ANSIBLE SYSTEM ROLE

You can integrate Microsoft SQL Server into an Active Directory (AD) to enable AD users to authenticate to SQL Server. By using the **microsoft.sql.server** Ansible system role, you can automate this process and remotely install and configure SQL Server accordingly. Note that you must still perform manual steps in AD and SQL Server after you run the playbook.

Depending on the RHEL version on the managed host, the version of SQL Server that you can install differs:

- RHEL 7.9: SQL Server 2017 and 2019
- RHEL 8: SQL Server 2017, 2019, and 2022
- RHEL 9.4 and later: SQL Server 2022

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.
- You installed the **ansible-collection-microsoft-sql** package or the **microsoft.sql** collection on the control node.
- The managed node has 2 GB or more RAM installed.
- The managed node uses one of the following versions: RHEL 7.9, RHEL 8, RHEL 9.4 or later.
- An AD domain is available in the network.
- A reverse DNS (RDNS) zone exists in AD, and it contains Pointer (PTR) resource records for each AD domain controller (DC).
- The managed host's network settings use an AD DNS server.
- The managed host can resolve the following DNS entries:
  - Both the hostnames and the fully-qualified domain names (FQDNs) of the AD DCs resolve to their IP addresses.
  - The IP addresses of the AD DCs resolve to their FQDNs.

Procedure

1. Store your sensitive variables in an encrypted file:
   a. Create the vault:

      ```
      $ ansible-vault create ~/vault.yml
      New Vault password: <vault_password>
      Confirm New Vault password: <vault_password>
      ```

   b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the ***<key>*: *<value>*** format:

      ```
      sa_pwd: <sa_password>
      sql_pwd: <SQL_AD_password>
      ad_admin_pwd: <AD_admin_password>
      ```

   c. Save the changes, and close the editor. Ansible encrypts the data in the vault.
2. Create a playbook file, for example **~/playbook.yml**, with the following content:

```
---
- name: Installing and configuring Microsoft SQL Server
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: SQL Server with AD authentication
      ansible.builtin.include_role:
        name: microsoft.sql.server
      vars:
        mssql_accept_microsoft_odbc_driver_for_sql_server_eula: true
        mssql_accept_microsoft_cli_utilities_for_sql_server_eula: true
```

```
    mssql_accept_microsoft_sql_server_standard_eula: true

    mssql_version: 2022
    mssql_password: "{{ sa_pwd }}"
    mssql_edition: Developer
    mssql_tcp_port: 1433
    mssql_manage_firewall: true

    mssql_ad_configure: true
    mssql_ad_join: true
    mssql_ad_sql_user: sqluser
    mssql_ad_sql_password: "{{ sql_pwd }}"
    ad_integration_realm: ad.example.com
    ad_integration_user: Administrator
    ad_integration_password: "{{ ad_admin_pwd }}"
```

The settings specified in the example playbook include the following:

mssql_ad_configure: true
: Enables authentication against AD.

mssql_ad_join: true
: Uses the **ad_integration** RHEL system role to join the managed node to AD. The role uses the settings from the **ad_integration_realm**, **ad_integration_user**, and **ad_integration_password** variables to join the domain.

mssql_ad_sql_user: *<username>*
: Sets the name of an AD account that the role should create in AD and SQL Server for administration purposes.

ad_integration_user: *<AD_user>*
: Sets the name of an AD user with privileges to join machines to the domain and to create the AD user specified in **mssql_ad_sql_user**.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/microsoft.sql-server/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --ask-vault-pass --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

5. Authorize AD users that should be able to authenticate to SQL Server. On the SQL Server, perform the following steps:

   a. Obtain a Kerberos ticket for the **Administrator** user:

   ```
   $ kinit Administrator@ad.example.com
   ```

   b. Authorize an AD user:

```
$ /opt/mssql-tools/bin/sqlcmd -S. -Q 'CREATE LOGIN [AD\<AD_user>] FROM WINDOWS;'
```

Repeat this step for every AD user who should be able to access SQL Server.

Verification

- On the managed node that runs SQL Server:
  a. Obtain a Kerberos ticket for an AD user:

  ```
  $ kinit <AD_user>@ad.example.com
  ```

  b. Use the **sqlcmd** utility to log in to SQL Server and run a query, for example:

  ```
  $ /opt/mssql-tools/bin/sqlcmd -S. -Q 'SELECT SYSTEM_USER'
  ```

Additional resources

- **/usr/share/ansible/roles/microsoft.sql-server/README.md** file
- Ansible vault