Red Hat Enterprise Linux 10

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: 2026-02-02

Red Hat Enterprise Linux 10 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

Legal Notice

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.

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 host. For more information, see Preparing a control node on RHEL 10 .

IMPORTANT

RHEL 10 contains **ansible-core** 2.16. This Ansible version supports managing RHEL 9 and RHEL 10 nodes.

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 10 control node

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

- **ad_integration**: Active Directory integration
- **aide**: Advanced Intrusion Detection Environment
- **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 Red Hat Lightspeed 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

Additional resources

- Red Hat Enterprise Linux (RHEL) system roles

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 10

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

- 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]# dnf 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 steps

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

Additional resources

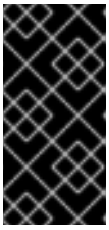
- How to register and subscribe a system to the Red Hat Customer Portal using subscription-manager (Red Hat Knowledgebase)

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 10 .
- 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 a managed host by using the Ansible **command** module:

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

If the command returns root, you configured **sudo** on the managed nodes correctly.

CHAPTER 3. ANSIBLE VAULT

You can use Ansible vault to encrypt sensitive data, such as passwords and API keys, in your playbooks.

Storing sensitive data 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.

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.

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 (RHEL) host to this AD. 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.

For example, if a host is joined to AD, AD users can then log in to RHEL and you can make services on the RHEL host available for authenticated AD users.

NOTE

The **ad_integration** role is for deployments using direct AD integration without an Identity Management (IdM) in Red Hat Enterprise Linux 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 for these nodes.
- 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.
- The managed node can establish connections to AD domain controllers by using the following ports:

Source Ports	Destination Port	Protocol	Service
1024 - 65535	53	UDP and TCP	DNS
1024 - 65535	389	UDP and TCP	LDAP
1024 - 65535	636	TCP	LDAPS
1024 - 65535	88	UDP and TCP	Kerberos
1024 - 65535	464	UDP and TCP	Kerberos password change requests
1024 - 65535	3268	TCP	LDAP Global Catalog

Source Ports	Destination Port	Protocol	Service
1024 - 65535	3269	TCP	LDAPS Global Catalog
1024 - 65535	123	UDP	NTP (if time synchronization is enabled)
1024 - 65535	323	UDP	NTP (if time synchronization is enabled)

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

- Ansible vault

CHAPTER 5. CONFIGURING THE GRUB 2 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 GRUB2 boot loader.

This role currently supports configuring the GRUB2 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 GRUB2 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 for these nodes.
- 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 GRUB2 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-6.12.0-0.el10_0.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 needed 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-6.12.0-0.el10_0.aarch64"
args="ro crashkernel=2G-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-6.12.0-0.el10_0.aarch64.img $tuned_initrd"
title="Red Hat Enterprise Linux (6.12.0-0.el10_0.aarch64) 10"
id="2c9ec787230141a9b087f774955795ab-6.12.0-0.el10_0.aarch64"
...
```

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 GRUB2 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 for these 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:

```
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 GRUB2 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
```

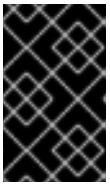
The settings specified in the example playbook include the following:

bootloader_password: *"{{ pwd }}"*

The variable ensures protection of boot parameters with a password.

bootloader_reboot_ok: *true*

The role detects that a reboot is needed 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 GRUB2 boot menu screen, press the **e** key for edit.
2. You will be prompted for a username and a 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.

Additional resources

- Ansible vault

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 GRUB2 boot loader menu in an automated fashion. This way you can efficiently update a period of time during which you can 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 for these nodes.

Procedure

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

```
---
- name: Configuration and management of GRUB2 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 GRUB2 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 GRUB2 boot menu screen.

   The highlighted entry will be executed automatically in 10s

   For how long this boot menu is displayed before GRUB2 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
```

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 GRUB2 boot loader entries in an automated fashion. This way you can quickly identify that your systems are set up to boot correctly, all entries point to the right kernels and initial RAM disk images.

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 for these nodes.

Procedure

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

```
---
- name: Configuration and management of GRUB2 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-6.12.el10_0.aarch64",
            "index": "1",
            "initrd": "/boot/initramfs-6.12.0.el10_0.aarch64.img $tuned_initrd",
            "kernel": "/boot/vmlinuz-6.12.0-0.el10_0.aarch64",
            "root": "/dev/mapper/rhel-root",
            "title": "Red Hat Enterprise Linux (6.12.0-0.el10_0.aarch64) 10"
        }
    ]
...
```

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.

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. 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 request a certificate from a CA.

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

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 Identity Management (IdM) in Red Hat Enterprise Linux certificate authority (CA).

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 for these nodes.
- 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
```

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.

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 for these nodes.

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
```

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 THE COCKPIT RHEL SYSTEM ROLE

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

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 for these nodes.

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_port: 9090
        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

- Requesting certificates from a CA and creating self-signed certificates by using RHEL system roles

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

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.

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.

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** cryptographic policy on your managed nodes.

The **FUTURE** 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

In some industries, 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 and software do 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 interoperability 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 for these nodes.

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 subpolicies. The specified base policy and subpolicies have to be available on the managed node. The default value is **null**, which means that the configuration is not changed and the **crypto_policies** RHEL system role only collects 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 the role variables and the cryptographic configuration options, see the **/usr/share/ansible/roles/rhel-system-roles.crypto_policies/README.md** file and the **update-crypto-policies(8)** and **crypto-policies(7)** manual pages 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 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.

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. You can automate the 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 for these nodes.

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
```

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

The **firewall** RHEL system role supports automating a reset of **firewalld** settings to their defaults. This efficiently removes insecure or unintentional firewall rules and simplifies 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 for these nodes.

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'
```

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 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 for these nodes.

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 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=
```

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

You can use the **firewall** RHEL system role to configure a zone to allow certain traffic. For example, you can configure that the **dmz** zone with the **enp1s0** interface allows HTTPS traffic to 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 for these nodes.

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:
```

10.4. CREATING A CUSTOM FIREWALLD SERVICE BY USING THE FIREWALL RHEL SYSTEM ROLE

In **firewalld**, a service is a named collection of rules that permit traffic for specific applications. Instead of manually managing individual ports and protocols, administrators can open up traffic by using a service name.

You can use the **firewall** RHEL system role to automate the creation of custom service files, making your firewall configurations simpler and more reusable.

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 for these nodes.

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: Create a firewalld service
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.firewall
      vars:
        firewall:
          service: custom_service
          short: A custom firewalld service
          description: >-
            A custom firewalld service that opens port 2222/tcp and
            the ports opened by the http and https firewalld services.
          port: 2222/tcp
          includes:
            - http
            - https
          state: present
          permanent: true
```

The settings specified in the example playbook include the following:

service: *<service_name>*

Sets the name of the service.

short: *<short_description>*

Sets a short description for the service.

description: *<description>*

Sets a long description for the service.

port: *<port>*/*<protocol>*

Defines the ports and protocols the service file should allow. To define multiple entries, use a YAML list.

includes: *<services>*

Optional: Defines other **firewalld** service files the service you want to create should include.

state: present

Adds the service. If the service already exists, the role modifies it as defined.

permanent: true

Enables changes in the permanent configuration of **firewalld**.

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

- Display the service definition you created:

```
# ansible managed-node-01.example.com -m ansible.builtin.command -a 'firewall-cmd --info-service=custom_service'
```

CHAPTER 11. CONFIGURING A HIGH-AVAILABILITY CLUSTER BY USING THE HA_CLUSTER SYSTEM ROLE

With the **ha_cluster** system role, you can configure and manage a system-roles cluster that uses the Pacemaker high availability cluster resource manager.

11.1. SPECIFYING AN INVENTORY FOR THE HA_CLUSTER RHEL SYSTEM ROLE

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

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
```

You can optionally configure watchdog and SBD devices for each node in an inventory. All SBD devices must be shared to and accessible from all nodes. Watchdog devices can be different for each node as well. For an example procedure that configures SBD node fencing in an inventory file, see Configuring a high availability cluster with SBD node fencing by using the ha_cluster variable.

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

You can use the **ha_cluster** RHEL system role to create Transport Layer Security (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.

The connection between cluster nodes is secured using 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**.

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 for these nodes.
- 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 10 .

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

- Ansible vault

- Requesting certificates using RHEL system roles

11.3. 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.

This example 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 for these nodes.
- 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 10 .

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
```

11.4. 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. You can use the **ha_cluster** RHEL system role to configure a cluster with 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 for these nodes.
- 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 10 .

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

- Configuring fencing in a Red Hat High Availability cluster

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

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 for these nodes.
- 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 10 .

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
```

11.6. CONFIGURING A HIGH AVAILABILITY CLUSTER WITH FENCING LEVELS

You can use the **ha_cluster** RHEL system role to configure high availability clusters with fencing levels. With 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.

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 for these nodes.
- 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 10 .

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

- Configuring fencing levels

11.7. CONFIGURING A HIGH AVAILABILITY CLUSTER WITH RESOURCE CONSTRAINTS USING SYSTEM ROLES

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 for these nodes.

- 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 10 .

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
```

11.8. CONFIGURING COROSYNC VALUES IN A HIGH AVAILABILITY CLUSTER USING RHEL SYSTEM ROLES

You can use the **ha_cluster** RHEL system role to configure Corosync values in high availability clusters.

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.

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 for these nodes.
- 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 10 .

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
          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
```

11.9. EXPORTING A CLUSTER CONFIGURATION TO CREATE A RHEL SYSTEM ROLE PLAYBOOK

You can use the **ha_cluster** RHEL system role to export the Corosync configuration of a cluster into **ha_cluster** variables that can be fed back to the role to recreate the same cluster.

If you did not use **ha_cluster** to create your cluster, or if you do not have access to the original playbook for the cluster, you can use this feature to build a new playbook for creating the cluster.

When you export a cluster's configuration by using the **ha_cluster** RHEL system role, not all of the variables are exported. You must manually modify the configuration to account for these variables.

The following variables are present in the export:

- **ha_cluster_cluster_present**
- **ha_cluster_start_on_boot**
- **ha_cluster_cluster_name**
- **ha_cluster_transport**
- **ha_cluster_totem**
- **ha_cluster_quorum**
- **ha_cluster_node_options** - Only the **node_name**, **corosync_addresses** and **pcs_address** options are present.
- **ha_cluster_enable_repos**
- **ha_cluster_enable_repos_resilient_storage**
- **ha_cluster_manage_firewall**
- **ha_cluster_manage_selinux**
- **ha_cluster_install_cloud_agents**
- **ha_cluster_pcs_permission_list**
- **ha_cluster_resource_primitives**
- **ha_cluster_resource_groups**

- **ha_cluster_resource_clones**
- **ha_cluster_resource_bundles**

The following variables are not present in the export:

- **ha_cluster_hacluster_password** - This is a mandatory variable for the role but it cannot be extracted from existing clusters.
- **ha_cluster_corosync_key_src**, **ha_cluster_pacemaker_key_src** and **ha_cluster_fence_virt_key_src** - These variables should contain paths to files with Corosync and Pacemaker keys. Since the keys themselves are not exported, these variables are not present in the export either. These keys should be unique for each cluster.
- **ha_cluster_regenerate_keys** - You should decide whether to use existing keys or to generate new ones.
- **ha_cluster_hacluster_qdevice_password** - Specifies the password for the **ha_cluster** user on a quorum device. This is a required setting when using a quorum device to avoid manual intervention. It cannot be extracted from existing clusters.
- **ha_cluster_fence_agent_packages** - A list of fence agent packages to install.
- **ha_cluster_extra_packages** - Specifies any extra packages to be installed on the cluster nodes.
- **ha_cluster_use_latest_packages** - When set to **true**, the system role will use the latest available packages for the cluster.
- **ha_cluster_pcsd_public_key_src**, **ha_cluster_pcsd_private_key_src** - These variables should contain paths to TLS certificate and private key for **pcsd**. Since the certificate and key themselves are not exported, these variables are not present in the export either.
- **ha_cluster_pcsd_certificates** - When this variable is set, the certificate RHEL system role is used internally to create the private key and certificate for **pcsd**. It cannot be extracted from existing clusters.

To export the current cluster configuration, run the **ha_cluster** RHEL system role and set **ha_cluster_export_configuration: true**. This triggers the export once the role finishes configuring a cluster or a **qnetd** host and stores it in the **ha_cluster_facts** variable.

By default, **ha_cluster_cluster_present** is set to **true** and **ha_cluster_qnetd.present** is set to **false**. These settings will reconfigure your cluster on the specified hosts, remove **qnetd** configuration from the specified hosts, and then export the configuration. To trigger the export without modifying an existing configuration, run the role with the following settings:

```
- hosts: node1
  vars:
    ha_cluster_cluster_present: null
    ha_cluster_qnetd: null
    ha_cluster_export_configuration: true

  roles:
    - linux-system-roles.ha_cluster
```

The following procedure:

- Exports the cluster configuration from cluster node **node1** into the **ha_cluster_facts** variable.
- Sets the **ha_cluster_cluster_present** and **ha_cluster_qnetd** variables to null to ensure that running this playbook does not modify the existing cluster configuration.
- Uses the Ansible debug module to display the content of **ha_cluster_facts**.
- Saves the contents of **ha_cluster_facts** to a file on the control node in a YAML format for you to write a playbook around it.

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 for these nodes.
- You have previously configured the high availability cluster with the configuration to export.
- You have created an inventory file on the control node, as described in Preparing a control node on RHEL 10.

Procedure

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

```
---
- name: Export high availability cluster configuration
  hosts: node1
  Tasks:
    - name: Export configuration that does not modify existing cluster
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.ha_cluster
      vars:
        ha_cluster_cluster_present: null
        ha_cluster_qnetd: null
        ha_cluster_export_configuration: true
    - name: Print ha_cluster_info_result variable
      ansible.builtin.debug:
        var: ha_cluster_facts
    - name: Save current cluster configuration to a file
      delegate_to: localhost
      ansible.builtin.copy:
        content: "{{ ha_cluster_facts | to_nice_yaml(sort_keys=false) }}"
        dest: /path/to/file
        mode: "0640"
```

The settings specified in the example playbook include the following:

hosts: node1

A node containing the cluster information to export.

ha_cluster_cluster_present: null

Setting to indicate that the cluster configuration will not be changed on the specified host.

ha_cluster_qnetd: null

Setting to indicate that the qnetd host configuration will not be changed on the specified host.

ha_cluster_export_configuration: true

A variable that determines whether to export the current cluster configuration and store it in the **ha_cluster_facts** variable, which is generated by the **ha_cluster_info** module.

ha_cluster_facts

A variable that contains the exported cluster configuration.

delegate_to: localhost

Specifies the control node as the location for the exported configuration file.

content: "{{ ha_cluster_facts | to_nice_yaml(sort_keys=false) }"}, dest: */path/to/file*, mode: "0640"

Copies the configuration file in a YAML format to */path/to/file*, setting the file permissions to 0640.
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.

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
```

4. After you have exported the current cluster configuration, you can write a playbook for your system using the variables you exported to */path/to/file* on the control node.
You must add the **ha_cluster_hacluster_password** variable, as it is a required variable but is not present in the export. Optionally, add the **ha_cluster_corosync_key_src**, **ha_cluster_pacemaker_key_src**, **ha_cluster_fence_virt_key_src**, and **ha_cluster_regenerate_keys** variables if your system requires them. These variables are never exported.

11.10. CONFIGURING A HIGH AVAILABILITY CLUSTER THAT IMPLEMENTS ACCESS CONTROL LISTS (ACLS) BY USING THE HA_CLUSTER RHEL SYSTEM ROLE

You can use the **ha_cluster** RHEL system role to configure high availability clusters with access control lists (ACLs). With ACLs, you can grant permission for specific local users other than user **hacluster** to manage a Pacemaker cluster.

A common use case for this feature is to restrict unauthorized users from accessing business-sensitive information.

By default, ACLs are not enabled. Consequently, any member of the group **haclient** on all nodes has full local read and write access to the cluster configuratioan. Users who are not members of **haclient** have no access. When ACLs are enabled, however, even users who are members of the **haclient** group have access only to what has been granted to that user by the ACLs. The **root** and **hacluster** user accounts always have full access to the cluster configuration, even when ACLs are enabled.

When you set permissions for local users with ACLs, you create a role which defines the permissions for that role. You then assign that role to a user. If you assign multiple roles to the same user, any deny permission takes precedence, then write, then read.

The following example procedure uses the **ha_cluster** RHEL system role to create in an automated fashion a high availability cluster that implements ACLs to control access to the cluster configuration.

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 for these nodes.
- 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 10 .

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: Configure a cluster with ACLs assigned
      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
        # To use an ACL role permission reference, the reference must exist in CIB.
        ha_cluster_resource_primitives:
          - id: not-for-operator
            agent: 'ocf:pacemaker:Dummy'
        # ACLs must be enabled (using the enable-acl cluster property) in order to be effective.
        ha_cluster_cluster_properties:
          - attrs:
              - name: enable-acl
                value: 'true'
        ha_cluster_acls:
          acl_roles:
            - id: operator
              description: HA cluster operator
              permissions:
                - kind: write
                  xpath: //crm_config//nvpair[@name='maintenance-mode']
                - kind: deny
                  reference: not-for-operator
            - id: administrator
              permissions:
                - kind: write
                  xpath: /cib
          acl_users:
            - id: alice
              roles:
                - operator
                - administrator
            - id: bob
              roles:
                - administrator
          acl_groups:
            - id: admins
              roles:
                - administrator
```

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 resources.

ha_cluster_cluster_properties: *<cluster properties>*

A list of sets of cluster properties for Pacemaker cluster-wide configuration.

ha_cluster_acls: *<dictionary>*

A dictionary of ACL role, user, and group values.

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

- Ansible vault

11.11. CONFIGURING A HIGH AVAILABILITY CLUSTER WITH SBD NODE FENCING BY USING THE HA_CLUSTER_NODE_OPTIONS VARIABLE

You can use the **ha_cluster** RHEL system role to configure high availability clusters with STONITH Block Device (SBD) fencing in an automated fashion.

You must configure a Red Hat high availability cluster with at least one fencing device to ensure the cluster-provided services remain available when a node in the cluster encounters a problem. If your environment does not allow for a remotely accessible power switch to fence a cluster node, you can configure fencing by using a STONITH Block Device (SBD). This device provides a node fencing mechanism for Pacemaker-based clusters through the exchange of messages by means of shared block storage. SBD integrates with Pacemaker, a watchdog device and, optionally, shared storage to arrange for nodes to reliably self-terminate when fencing is required.

You can use the **ha_cluster** RHEL system role to configure SBD fencing in an automated fashion. With **ha_cluster**, you can configure watchdog and SBD devices on a node-to-node basis by using one of two variables:

- **ha_cluster_node_options**: This is a single variable you define in a playbook file. It is a list of dictionaries where each dictionary defines options for one node.
- **ha_cluster**: A dictionary that defines options for one node only. You configure the **ha_cluster** variable in an inventory file. To set different values for each node, you define the variable separately for each node.

If both the **ha_cluster_node_options** and **ha_cluster** variables contain SBD options, those in **ha_cluster_node_options** have precedence.

This example procedure uses the **ha_cluster_node_options** variable in a playbook file to configure node addresses and SBD options on a per-node basis. For an example procedure that uses the **ha_cluster** variable in an inventory file, see Configuring a high availability cluster with SBD node fencing by using the ha_cluster variable.

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 for these nodes.
- 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 10 .

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: Configure a cluster with SBD fencing
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.ha_cluster
      vars:
        my_sbd_devices:
          # This variable is indirectly used by various variables of the ha_cluster RHEL system role.
          # Its purpose is to define SBD devices once so they do not need
          # to be repeated several times in the role variables.
          - /dev/disk/by-id/000001
          - /dev/disk/by-id/000002
          - /dev/disk/by-id/000003
        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_sbd_enabled: true
        ha_cluster_sbd_options:
          - name: delay-start
            value: 'no'
          - name: startmode
            value: always
          - name: timeout-action
            value: 'flush,reboot'
          - name: watchdog-timeout
            value: 30
        ha_cluster_node_options:
          - node_name: node1
            sbd_watchdog_modules:
              - iTCO_wdt
            sbd_watchdog_modules_blocklist:
              - ipmi_watchdog
            sbd_watchdog: /dev/watchdog1
            sbd_devices: "{{ my_sbd_devices }}"
          - node_name: node2
            sbd_watchdog_modules:
              - iTCO_wdt
            sbd_watchdog_modules_blocklist:
              - ipmi_watchdog
            sbd_watchdog: /dev/watchdog1
            sbd_devices: "{{ my_sbd_devices }}"
```

```
    # Best practice for setting 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:
              - name: devices
                value: "{{ my_sbd_devices | join(',') }}"
              - name: pcmk_delay_base
                value: 30
```

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_sbd_enabled: true

A variable that determines whether the cluster can use the SBD node fencing mechanism.

ha_cluster_sbd_options: *<sbd options>*

A 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 on your system.

ha_cluster_node_options: *<node options>*

A variable that defines settings which vary from one cluster node to another. You can configure the following SBD and watchdog items:

- **sbd_watchdog_modules** - Modules to be loaded, which create **/dev/watchdog*** devices.
- **sbd_watchdog_modules_blocklist** - Watchdog kernel modules to be unloaded and blocked.
- **sbd_watchdog** - Watchdog device to be used by SBD.
- **sbd_devices** - Devices to use for exchanging SBD messages and for monitoring. Always refer to the devices using the long, stable device name (**/dev/disk/by-id/**).

ha_cluster_cluster_properties: *<cluster properties>*

A list of sets of cluster properties for Pacemaker cluster-wide configuration.

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 resources.

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

- Exploring RHEL High Availability's Components - sbd and fence_sbd
- Design Guidance for RHEL High Availability Clusters - sbd Considerations
- Support Policies for RHEL High Availability Clusters - sbd and fence_sbd
- Exploring RHEL High Availability's Components - sbd and fence_sbd (Red Hat Knowledgebase)
- Design Guidance for RHEL High Availability Clusters - sbd Considerations (Red Hat Knowledgebase)
- Support Policies for RHEL High Availability Clusters - sbd and fence_sbd (Red Hat Knowledgebase)
- Ansible vault

11.12. CONFIGURING A HIGH AVAILABILITY CLUSTER WITH SBD NODE FENCING BY USING THE HA_CLUSTER VARIABLE

You can use the **ha_cluster** RHEL system role to configure high availability clusters with STONITH Block Device (SBD) fencing in an automated fashion.

You must configure a Red Hat high availability cluster with at least one fencing device to ensure the cluster-provided services remain available when a node in the cluster encounters a problem. If your environment does not allow for a remotely accessible power switch to fence a cluster node, you can configure fencing by using a SBD. This device provides a node fencing mechanism for Pacemaker-based clusters through the exchange of messages by means of shared block storage. SBD integrates with Pacemaker, a watchdog device and, optionally, shared storage to arrange for nodes to reliably self-terminate when fencing is required.

With **ha_cluster**, you can configure watchdog and SBD devices on a node-to-node basis by using one of two variables:

- **ha_cluster_node_options**: This is a single variable you define in a playbook file. It is a list of dictionaries where each dictionary defines options for one node.
- **ha_cluster**: A dictionary that defines options for one node only. You configure the **ha_cluster** variable in an inventory file. To set different values for each node, you define the variable separately for each node.

If both the **ha_cluster_node_options** and **ha_cluster** variables contain SBD options, those in **ha_cluster_node_options** have precedence.

If both the **ha_cluster_node_options** and **ha_cluster** variables contain SBD options, those in **ha_cluster_node_options** have precedence.`

The following example procedure uses the **ha_cluster** system role to create a high availability cluster with SBD fencing. This example procedure uses the **ha_cluster** variable in an inventory file to configure node addresses and SBD options on a per-node basis. For an example procedure that uses the **ha_cluster_node_options** variable in a playbook file, see Configuring a high availability cluster with SBD node fencing by using the **ha_cluster_nodes_options** variable.

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 for these nodes.
- 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 10 .

Procedure

1. Create an inventory file for your cluster that configures watchdog and SBD devices for each node by using the **ha_cluster** variable, as in the following exampla;e

```
all:
  hosts:
    node1:
      ha_cluster:
        sbd_watchdog_modules:
          - iTCO_wdt
```

```
      sbd_watchdog_modules_blocklist:
        - ipmi_watchdog
      sbd_watchdog: /dev/watchdog1
      sbd_devices:
        - /dev/disk/by-id/000001
        - /dev/disk/by-id/000001
        - /dev/disk/by-id/000003
    node2:
      ha_cluster:
        sbd_watchdog_modules:
          - iTCO_wdt
        sbd_watchdog_modules_blocklist:
          - ipmi_watchdog
        sbd_watchdog: /dev/watchdog1
        sbd_devices:
          - /dev/disk/by-id/000001
          - /dev/disk/by-id/000002
          - /dev/disk/by-id/000003
```

The SBD and watchdog settings specified in the example inventory include the following:

sbd_watchdog_modules

Watchdog kernel modules to be loaded, which create **/dev/watchdog*** devices.

sbd_watchdog_modules_blocklist

Watchdog kernel modules to be unloaded and blocked.

sbd_watchdog

Watchdog device to be used by SBD.

sbd_devices

Devices to use for exchanging SBD messages and for monitoring. Always refer to the devices using the long, stable device name (**/dev/disk/by-id/**).

For general information about creating an inventory file, see Preparing a control node on RHEL 10.

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:

   ```
   cluster_password: <cluster_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**, as in the following example. Since you have specified the SBD and watchog variables in an inventory, you do not need to include them in the playbook.

```
---
- name: Create a high availability cluster
  hosts: node1 node2
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Configure a cluster with sbd fencing devices configured in an inventory file
      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_sbd_enabled: true
        ha_cluster_sbd_options:
          - name: delay-start
            value: 'no'
          - name: startmode
            value: always
          - name: timeout-action
            value: 'flush,reboot'
          - name: watchdog-timeout
            value: 30
        # Best practice for setting 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
                  # this only works if all nodes have the same sbd_devices
                  - name: devices
                    value: "{{ ha_cluster.sbd_devices | join(',') }}"
                  - name: pcmk_delay_base
                    value: 30
```

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_sbd_enabled: true

A variable that determines whether the cluster can use the SBD node fencing mechanism.

ha_cluster_sbd_options: *sbd options*

A 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 on your system.

ha_cluster_cluster_properties: *cluster properties*

A list of sets of cluster properties for Pacemaker cluster-wide configuration.

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 resources.

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.

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

- Exploring RHEL High Availability's Components - sbd and fence_sbd (Red Hat Knowledgebase)
- Design Guidance for RHEL High Availability Clusters - sbd Considerations (Red Hat Knowledgebase)
- Support Policies for RHEL High Availability Clusters - sbd and fence_sbd (Red Hat Knowledgebase)
- Ansible vault

11.13. CONFIGURING A PLACEMENT STRATEGY FOR A HIGH AVAILABILITY CLUSTER BY USING THE RHEL HA_CLUSTER RHEL SYSTEM ROLE

You can use the **ha_cluster** RHEL system role to create a high availability cluster in an automated fashion that configures utilization attributes to define a placement strategy.

A Pacemaker cluster allocates resources according to a resource allocation score. By default, if the resource allocation scores on all the nodes are equal, Pacemaker allocates the resource to the node with the smallest number of allocated resources. If the resources in your cluster use significantly different

proportions of a node's capacities, such as memory or I/O, the default behavior may not be the best strategy for balancing your system's workload. In this case, you can customize an allocation strategy by configuring utilization attributes and placement strategies for nodes and resources.

For detailed information about configuring utilization attributes and placement strategies, see Configuring a node placement strategy .

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 for these nodes.
- 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 10 .

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: Configure a cluster with utilization 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_cluster_properties:
          - attrs:
              - name: placement-strategy
                value: utilization
        ha_cluster_node_options:
          - node_name: node1
            utilization:
              - attrs:
                  - name: utilization1
                    value: 1
                  - name: utilization2
                    value: 2
          - node_name: node2
            utilization:
              - attrs:
                  - name: utilization1
                    value: 3
                  - name: utilization2
                    value: 4
        ha_cluster_resource_primitives:
          - id: resource1
            agent: 'ocf:pacemaker:Dummy'
            utilization:
              - attrs:
                  - name: utilization1
                    value: 2
                  - name: utilization2
                    value: 3
```

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_cluster_properties: *<cluster properties>*

List of sets of cluster properties for Pacemaker cluster-wide configuration. For utilization to have an effect, the **placement-strategy** property must be set and its value must be different from the value **default**.

`**ha_cluster_node_options:** *<node options>*

A variable that defines various settings which vary from cluster node to cluster node.

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 resources.
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

- Ansible vault

11.14. CONFIGURING ALERTS FOR A HIGH AVAILABILITY CLUSTER BY USING THE HA_CLUSTER RHEL SYSTEM ROLE

You can use the **ha_cluster** RHEL system role to configure alerts for high availability clusters.

When a Pacemaker event occurs, such as a resource or a node failure or a configuration change, you may want to take some external action. For example, you may want to send an email message or log to a file or update a monitoring system.

You can configure your system to take an external action by using alert agents. These are external programs that the cluster calls in the same manner as the cluster calls resource agents to handle resource configuration and operation. The cluster passes information about the event to the agent through environment variables.

NOTE

The **ha_cluster** RHEL system role configures the cluster to call external programs to handle alerts. However, you must provide these programs and distribute them to cluster nodes.

For more detailed information about alert agents, see Triggering scripts for cluster events.

This example procedure uses the **ha_cluster** RHEL system role to create a high availability cluster in an automated fashion that configures a Pacemaker alert.

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 for these nodes.
- 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 10 .

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: Configure a cluster with alerts
      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_alerts:
      - id: alert1
        path: /alert1/path
        description: Alert1 description
        instance_attrs:
          - attrs:
              - name: alert_attr1_name
                value: alert_attr1_value
        meta_attrs:
          - attrs:
              - name: alert_meta_attr1_name
                value: alert_meta_attr1_value
        recipients:
          - value: recipient_value
            id: recipient1
            description: Recipient1 description
            instance_attrs:
              - attrs:
                  - name: recipient_attr1_name
                    value: recipient_attr1_value
            meta_attrs:
              - attrs:
                  - name: recipient_meta_attr1_name
                    value: recipient_meta_attr1_value
```

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_alerts: *<alert definitions>*

A variable that defines Pacemaker alerts.

- **id** - ID of an alert.
- **path** - Path to the alert agent executable.
- **description**- Description of the alert.
- **instance_attrs** - List of sets of the alert's instance attributes. Currently, only one set is supported, so the first set is used and the rest are ignored.
- **meta_attrs** - List of sets of the alert's meta attributes. Currently, only one set is supported, so the first set is used and the rest are ignored.

- **recipients** - List of alert's recipients.
- **value**- Value of a recipient.
- **id** - ID of the recipient.
- **description** - Description of the recipient.
- **instance_attrs** -List of sets of the recipient's instance attributes. Currently, only one set is supported, so the first set is used and the rest are ignored.
- **meta_attrs** - List of sets of the recipient's meta attributes. Currently, only one set is supported, so the first set is used and the rest are ignored.

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

- Ansible vault

11.15. CONFIGURING A HIGH AVAILABILITY CLUSTER WITH A QUORUM DEVICE BY USING RHEL SYSTEM ROLES

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. Use a quorum device 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.

11.15.1. Configuring a quorum device

You can use the **ha_cluster** RHEL system role to configure a quorum device for high availability clusters. 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 for these nodes.
- 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 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 10 .

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.
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-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
```

11.15.2. Configuring a cluster to use a quorum device

You can use the **ha_cluster** RHEL system role to configure a cluster with a quorum device.

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 for these nodes.
- 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 10 .

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.
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.

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
```

11.16. CONFIGURING A HIGH AVAILABILITY CLUSTER WITH NODE ATTRIBUTES BY USING RHEL SYSTEM ROLES

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 for these nodes.
- 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 10 .

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

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

You can use the **ha_cluster** RHEL system role to configure an Apache HTTP server in a high availability cluster.

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 Knowledgebase article 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 for these nodes.
- 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 10 .
- You have configured an Logical Volume Manager (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
```

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
```

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**.

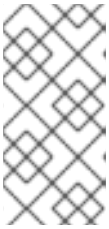
```
[root@z1 ~]# pcs status
Cluster name: my_cluster
```

```
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 website 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 .

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 for these nodes.

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
```

CHAPTER 13. CONFIGURING AUTOMATIC CRASH DUMPS BY USING RHEL SYSTEM ROLES

To manage **kdump** by using Ansible, you can use the **kdump** role, which is one of the RHEL system roles available in RHEL 10. 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 for these nodes.

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'
```

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 at once.

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 roles delivered over RHEL channels are available to RHEL customers as an RPM package in the default AppStream repository. RHEL system roles 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, by using the **kernel_settings** role, you can configure:

- Transparent hugepages to increase performance by reducing the overhead of managing smaller pages.
- The largest packet sizes are to be transmitted over the network with the loopback interface.
- Limits on files, which can 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 for these nodes.

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.

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 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'
```

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 **journald** 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.

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 for these nodes.

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 and more information about **rsyslog**, see the **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** file and **rsyslog.conf(5)** and **syslog(3)** manual pages 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**.

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

You can use the **logging** RHEL system role to configure centralized log management across multiple systems. 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 in to 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.
- 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 .

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 for these nodes.

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** are 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 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 the role variables and more information about **rsyslog**, see the **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** file and the **rsyslog.conf(5)** and **syslog(3)** manual pages 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**.

15.3. USING THE LOGGING RHEL SYSTEM ROLE WITH TLS

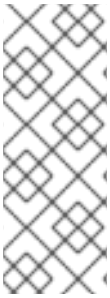
You can use the **logging** RHEL system role to configure a secure transfer of log messages, where one or more clients take logs from the **systemd-journal** service and transfer them to a remote server while using TLS.

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 by using TLS encryption.

The role 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 for these nodes.
- The managed nodes are enrolled in an IdM domain.
- If the logging server you want to configure on the managed node runs RHEL 9.2 or later and the FIPS mode is enabled, clients must either support the Extended Master Secret (EMS) extension or use TLS 1.3. TLS 1.2 connections without EMS fail. For more information, see the Red Hat Knowledgebase solution TLS extension "Extended Master Secret" enforced .

Procedure

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

```
---
- name: Configure remote logging solution by 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. The 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. The 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. The 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 the role variables and more information about **rsyslog**, see the **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** file and the **rsyslog.conf(5)** and **syslog(3)** manual pages 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
```

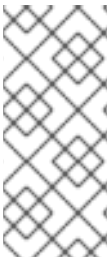
Additional resources

- Requesting certificates from a CA and creating self-signed certificates by using RHEL system roles

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 by using TLS encryption.

The role 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 for these nodes.
- The managed nodes are enrolled in an IdM domain.
- If the logging server you want to configure on the managed node runs RHEL 9.2 or later and FIPS mode is enabled, clients must either support the Extended Master Secret (EMS) extension

or use TLS 1.3. TLS 1.2 connections without EMS fail. For more information, see the Red Hat Knowledgebase solution TLS extension "Extended Master Secret" enforced .

Procedure

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

```
---
- name: Configure remote logging solution by 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]
```

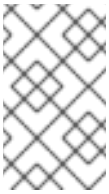
The settings specified in the example playbook include the following:

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. The 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. The 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. The 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 the role variables and more information about **rsyslog**, see the **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** file and the **rsyslog.conf(5)** and **syslog(3)** manual pages 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

- Requesting certificates from a CA and creating self-signed certificates by using RHEL system roles

15.4. USING THE LOGGING RHEL SYSTEM ROLES WITH RELP

You can use the **logging** RHEL system role to configure Reliable Event Logging Protocol (RELP) between a RELP client and RELP server.

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.

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.

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 for these nodes.

Procedure

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

```
---
- name: Configure client-side of the remote logging solution by 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 the local path on the control node to the specific path of the managed node.

ca_cert

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

cert

Represents the path to certificate. The 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. The 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 the role variables and more information about **rsyslog**, see the **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** file and the **rsyslog.conf(5)** and **syslog(3)** manual pages 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
```

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.

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 for these nodes.

Procedure

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

```
---
- name: Configure server-side of the remote logging solution by 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 the local path on the control node to the specific path of the managed node.

ca_cert

Represents the path to CA certificate. The 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. The 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. The 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 the role variables and more information about **rsyslog**, see the **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** file and the **rsyslog.conf(5)** and **syslog(3)** manual pages 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
```

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

16.1. CONFIGURING PERFORMANCE CO-PILOT BY USING THE METRICS 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 for these nodes.

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 .

16.2. CONFIGURING PERFORMANCE CO-PILOT WITH AUTHENTICATION BY USING THE METRICS RHEL SYSTEM ROLE

You can use the **metrics** RHEL system role to remotely configure Performance Co-Pilot (PCP) with authentication on multiple hosts.

You can enable authentication in 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.

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 for these 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:

```
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

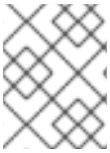
- Optional: Configure Grafana to monitor PCP hosts and visualize metrics .

Additional resources

- 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 configured Performance Co-Pilot (PCP) on multiple hosts, you can use Grafana to visualize the metrics for these hosts. By using the **metrics** RHEL system role, you can automate the process of setting up Grafana, the PCP plug-in, and the configuration of the data sources.



NOTE

If you use the **metrics** role to install Grafana on a host, the role also automatically installs 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 for these nodes.
- 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 **Menu → Apps → Performance Co-Pilot → 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 Valkey data source. This requires that you set **metrics_query_service: true** in the playbook.

Additional resources

- Ansible vault

16.4. CONFIGURING WEB HOOKS IN PERFORMANCE CO-PILOT BY USING THE METRICS RHEL SYSTEM ROLE

The Performance Co-Pilot (PCP) suite contains the performance metrics inference engine (PMIE) service. This service evaluates performance rules in real time. For example, you can use the default rules to detect excessive swap activities.

You can configure a host as a central PCP management site that collects the monitoring data from multiple PCP nodes. If a rule matches, this central host sends a notification to a web hook to notify other services. For example, the web hook can trigger Event-Driven Ansible to run on Ansible Automation Platform template or playbook on the host that had caused the event.

By using the **metrics** RHEL system role, you can automate the configuration of a central PCP management host that notifies a web hook.

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 for these nodes.
- PCP is configured for remote access on the hosts you want to monitor .
- The host on which you want to configure PMIE can access port 44321 on the PCP nodes you plan to monitor.

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 PMIE web hooks
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.metrics
      vars:
        metrics_manage_firewall: true
        metrics_retention_days: 7
        metrics_monitored_hosts:
          - pcp-node-01.example.com
          - pcp-node-02.example.com
        metrics_webhook_endpoint: "https://<webserver>:<port>/<endpoint>"
```

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_monitored_hosts: *<list_of_hosts>*

Specifies the hosts to observe.

metrics_webhook_endpoint: *<URL>*

Sets the web hook endpoint to which the performance metrics inference engine (PMIE) sends notifications about detected performance issues. By default, these issues are logged to the local system only.

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

1. Check the configuration summary on **managed-node-node-01.example.com**:

```
# ansible managed-node-01.example.com -m command -a 'pcp summary'
Performance Co-Pilot configuration on managed-node-01.example.com:

 platform: Linux managed-node-node-01.example.com 6.12.el10_0.x86_64 #1 SMP
PREEMPT_DYNAMIC Fri Feb 23 01:51:18 EST 2024 x86_64
 hardware: 8 cpus, 1 disk, 1 node, 1773MB RAM
 timezone: CEST-2
 services: pmcd pmproxy
 pmcd: Version 6.2.0-1, 12 agents, 6 clients
 pmda: root pmcd proc pmproxy xfs linux nfsclient mmv kvm jbd2
       dm openmetrics
 pmlogger: primary logger: /var/log/pcp/pmlogger/managed-node-node-
01.example.com/20240510.16.25
           pcp-node-01.example.com: /var/log/pmlogger/pcp-node-
01.example.com/20240510.16.25
           pcp-node-02.example.com: /var/log/pmlogger/pcp-node-
02.example.com/20240510.16.25
 pmie: primary engine: /var/log/pcp/pmie/managed-node-node-01.example.com/pmie.log
       pcp-node-01.example.com: : /var/log/pcp/pmie/pcp-node-01.example.com/pmie.log
       pcp-node-02.example.com: : /var/log/pcp/pmie/pcp-node-02.example.com/pmie.log
```

The last three lines confirm that PMIE is configured to monitor three systems.

Additional resources

- Automate performance management with Performance Co-Pilot by using Event-Driven Ansible

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 by using Clevis and Tang.

The **rhel-system-roles** package contains these system roles, the related examples, and 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 for these nodes.

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 Network-Bound Disk Encryption (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
```

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 the passphrase. After removing the passphrase, you can then 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 for these nodes.
- 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 your 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
…
```

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 Network-Bound Disk Encryption (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 for these nodes.
- 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

- Looking forward to Linux network configuration in the initial ramdisk (initrd)

CHAPTER 18. CONFIGURING NETWORK SETTINGS BY USING RHEL SYSTEM ROLES

Administrators can automate network-related configuration and management tasks by using the **network** RHEL system role.

18.1. CONFIGURING AN ETHERNET CONNECTION WITH A STATIC IP ADDRESS BY USING THE NETWORK RHEL SYSTEM ROLE WITH AN INTERFACE NAME

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.

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.

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 for these nodes.
- 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"
            ]
        },
...
```

18.2. CONFIGURING AN ETHERNET CONNECTION WITH A STATIC IP ADDRESS BY USING THE NETWORK RHEL SYSTEM ROLE WITH A DEVICE PATH

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.

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.

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 for these nodes.
- 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"
            ]
        },
...
```

18.3. CONFIGURING AN ETHERNET CONNECTION WITH A DYNAMIC IP ADDRESS BY USING THE NETWORK RHEL SYSTEM ROLE WITH AN INTERFACE NAME

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.

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.

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 for these nodes.
- A physical or virtual Ethernet device exists in the server's 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"
            ]
        },
...
```

18.4. CONFIGURING AN ETHERNET CONNECTION WITH A DYNAMIC IP ADDRESS BY USING THE NETWORK RHEL SYSTEM ROLE WITH A DEVICE PATH

By using the **network** RHEL system role, you can configure an Ethernet connection to retrieve its IP addresses, gateways, and DNS settings from a DHCP server and IPv6 stateless address autoconfiguration (SLAAC). The role can assign the profile by the device's 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.

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 for these nodes.
- 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"
            ]
        },
...
```

18.5. CONFIGURING A STATIC ETHERNET CONNECTION WITH 802.1X NETWORK AUTHENTICATION BY USING THE NETWORK RHEL SYSTEM ROLE

By using the **network** RHEL system role, you can automate setting up Network Access Control (NAC) on remote hosts. You can define authentication details for clients in a playbook to ensure only authorized clients can access the network.

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 for these nodes.
- 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

- Ansible vault

18.6. CONFIGURING A WIFI CONNECTION WITH 802.1X NETWORK AUTHENTICATION BY USING THE NETWORK RHEL SYSTEM ROLE

By using the **network** RHEL system role, you can automate setting up Network Access Control (NAC) on remote hosts. You can define authentication details for clients in a playbook to ensure only authorized clients can access the network.

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 for these nodes.

- The network supports 802.1X network authentication.
- You installed the **wpa_supplicant** package on the managed node.
- DHCP is available in the network of the managed node.
- The following files required for TLS authentication exist on the control node:
  - The client key is stored in the **/srv/data/client.key** file.
  - The client certificate is stored in the **/srv/data/client.crt** file.
  - The 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 a wifi 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: 0400

    - 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: Wifi connection profile with dynamic IP address settings and 802.1X
  ansible.builtin.import_role:
    name: redhat.rhel_system_roles.network
  vars:
    network_connections:
      - name: Wifi connection profile with dynamic IP address settings and 802.1X
        interface_name: wlp1s0
        state: up
        type: wireless
        autoconnect: yes
        ip:
          dhcp4: true
          auto6: true
        wireless:
          ssid: "Example-wifi"
          key_mgmt: "wpa-eap"
        ieee802_1x:
          identity: <user_name>
          eap: tls
          private_key: "/etc/pki/tls/private/client.key"
          private_key_password: "{{ pwd }}"
          private_key_password_flags: none
          client_cert: "/etc/pki/tls/certs/client.crt"
          ca_cert: "/etc/pki/ca-trust/source/anchors/ca.crt"
          domain_suffix_match: "example.com"
          network_allow_restart: true
```

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
```

Additional resources

- Ansible vault

18.7. CONFIGURING A NETWORK BOND BY USING THE NETWORK RHEL SYSTEM ROLE

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.

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.

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 for these nodes.
- 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.

18.8. CONFIGURING VLAN TAGGING BY USING THE NETWORK RHEL SYSTEM ROLE

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.

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.

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 for these nodes.

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
```

e 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
    ...
```

18.9. CONFIGURING A NETWORK BRIDGE BY USING THE NETWORK RHEL SYSTEM ROLE

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.

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.

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 for these nodes.
- 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
```

18.10. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION BY USING THE NETWORK RHEL SYSTEM ROLE

By using the **network** RHEL system role, you can automate setting the default gateway in a NetworkManager connection profile. With this method, you can remotely configure the default gateway on 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 for these nodes.

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
```

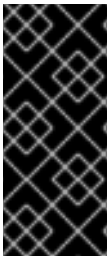
Verification

- 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",
    ...
}
...
```

18.11. CONFIGURING A STATIC ROUTE BY USING THE NETWORK RHEL SYSTEM ROLE

You can use the **network** RHEL system role to configure static routes.



IMPORTANT

When you run a play that uses the **network** RHEL system role and if the setting values do not match the values specified in the play, the role overrides the existing connection profile with the same name. To prevent resetting these values to their defaults, always specify the whole configuration of the network connection profile in the play, even if the configuration, for example the IP configuration, already exists.

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 for these nodes.

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: Configure an Ethernet connection with static IP and additional routes
      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:
              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
```

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

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

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 nodes:

a. Display the IPv4 routes:

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

b. Display the IPv6 routes:

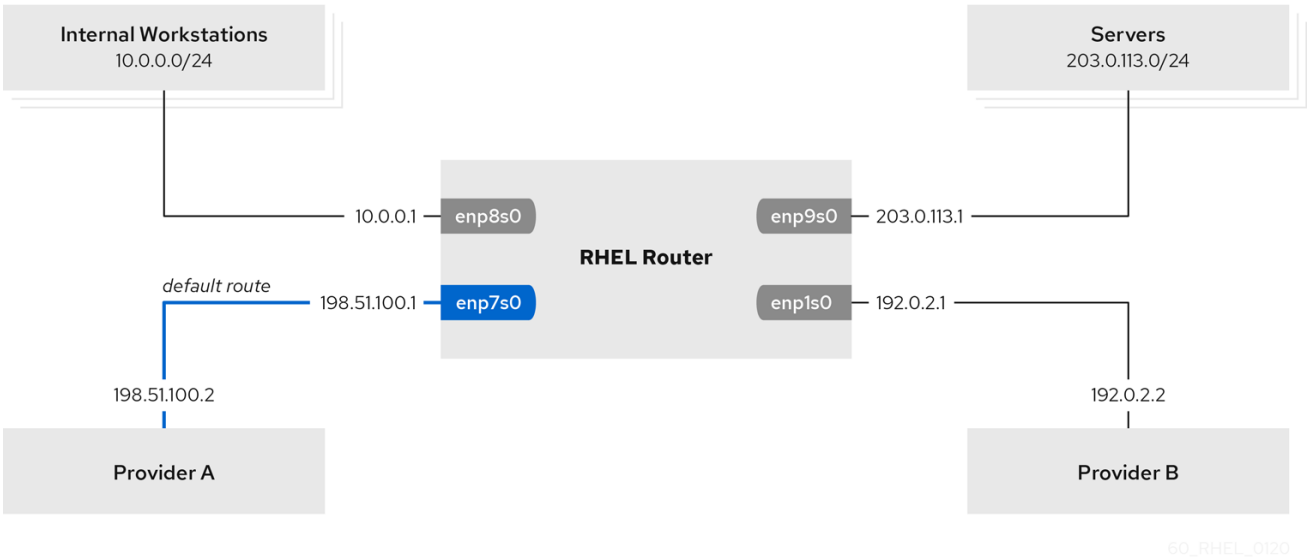
```
# ip -6 route
...
2001:db8:2::/64 via 2001:db8:1::10 dev enp7s0 metric 1024 pref medium
```

18.12. 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. By using the **network** RHEL system role, you can automate the creation of the connection profiles, including routing tables and rules.

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.

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 for these nodes.
- 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
```

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:

```
# dnf 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  _gateway (10.0.0.1)     0.337 ms  0.260 ms  0.223 ms
 2  192.0.2.1 (192.0.2.1)   0.884 ms  1.066 ms  1.248 ms
 ...
```

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

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

```
# dnf 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  _gateway (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
default 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
  ...
```

18.13. CONFIGURING AN ETHTOOL OFFLOAD FEATURE BY USING THE NETWORK RHEL SYSTEM ROLE

You can use the **network** RHEL system role to automate configuring TCP offload engine (TOE) to offload processing certain operations to the network controller. TOE improves the network throughput.

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 for these nodes.

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",
  ...
            }
...
```

18.14. CONFIGURING AN ETHTOOL COALESCE SETTING BY USING THE NETWORK RHEL SYSTEM ROLE

Interrupt coalescing collects network packets and generates a single interrupt for multiple packets. This reduces interrupt load and maximizes throughput. You can automate the configuration of these settings in the NetworkManager connection profile by using the **network** RHEL system role.

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 for these nodes.

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
...
```

18.15. 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 for these nodes.
- 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
```

18.16. CONFIGURING AN IPOIB CONNECTION BY USING THE NETWORK RHEL SYSTEM ROLE

To configure IP over InfiniBand (IPoIB), create a NetworkManager connection profile. You can automate this process by using the **network** RHEL system role and remotely configure connection profiles on 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 for these nodes.
- An InfiniBand device named **mlx5_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 mlx5_ib0
          - name: mlx5_ib0
            interface_name: mlx5_ib0
            type: infiniband

          # IPoIB device mlx5_ib0.8002 on top of mlx5_ib0
          - name: mlx5_ib0.8002
            type: infiniband
            autoconnect: yes
            infiniband:
              p_key: 0x8002
              transport_mode: datagram
            parent: mlx5_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 **mlx5_ib0.8002** device:

```
# ansible managed-node-01.example.com -m command -a 'ip address show
mlx5_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 **mlx5_ib0.8002** device:

```
# ansible managed-node-01.example.com -m command -a 'cat
/sys/class/net/mlx5_ib0.8002/pkey'
managed-node-01.example.com | CHANGED | rc=0 >>
0x8002
```

3. Display the mode of the **mlx5_ib0.8002** device:

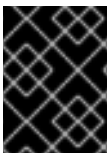
```
# ansible managed-node-01.example.com -m command -a 'cat
/sys/class/net/mlx5_ib0.8002/mode'
managed-node-01.example.com | CHANGED | rc=0 >>
datagram
```

18.17. 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:

Playbook with state configurations

```
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
```

Regular playbook

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

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

Playbook with state configurations

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

Regular playbook

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

CHAPTER 19. MANAGING CONTAINERS BY USING RHEL SYSTEM ROLES

The **podman** RHEL system roles manage containers on Red Hat Enterprise Linux systems. This role uses Ansible to configure **Podman**, manage container lifecycles, and even deploy containerized applications as systemd services.

19.1. CONFIGURING IMAGE REGISTRY MANAGEMENT FOR PODMAN AND OTHER CONTAINER TOOLS

With the **podman** RHEL system role, you can automate the Podman management, including registry configuration, across multiple RHEL systems. Instead of manually editing files, you define your desired registry configuration in an Ansible playbook.

The **podman** RHEL system role uses the **podman_registries_conf** variable, which accepts a dictionary containing the registry settings. The role then creates a drop-in file, for example, in the **/etc/containers/registries.conf.d/** to apply your configuration, following best practices for managing system configurations.

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 for these nodes.

Procedure

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

```
---
- name: Configure Podman registries with RHEL system roles
  hosts: managed-node-01.example.com
  vars:
    podman_registries_conf:
      unqualified-search-registries:
        - "registry.access.redhat.com"
        - "docker.io"
        - "my-company-registry.com"
      registry:
        - location: "my-company-registry.com"
        - location: "my-local-registry:5000"
          insecure: true
  tasks:
    - name: Include the podman system role
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.podman
```

The settings specified in the example playbook include the following:

- **unqualified-search-registries**: Extends the list of registries Podman searches when you use a short image name (for example, **podman pull <my-image>**). Podman searches for images in my-company-registry.com after the default registries.
- **[registry]**: Defines specific properties for a given registry. For example, you can enable an insecure connection by setting **insecure=true** to a local registry running at my-local-registry:5000.

The **podman_use_new_toml_formatter** variable generates TOML-compliant configuration files that are compatible with Podman. This variable enhances the Podman role by supporting all TOML features, including tables and inline tables, through a true TOML formatter instead of the Jinja template used previously.

The new formatter is disabled by default to maintain compatibility with the previous formatter's behavior. To enable the new formatter, set **podman_use_new_toml_formatter: true** in your configuration:

```
podman_use_new_toml_formatter: true
podman_containers_conf:
  containers:
    annotations:
      - environment=production
      - status=tier2
```

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. Run the **podman info** command on the host:

```
$ ansible managed-node-01.example.com -m command -a 'podman info'
```

2. Verify the registeries section:

```
registries:
  my-company-registry.com:
    Blocked: false
    Insecure: false
    Location: my-company-registry.com
    MirrorByDigestOnly: false
    Mirrors: null
    Prefix: my-company-registry.com
    PullFromMirror: ""
  my-local-registry:5000:
    Blocked: false
```

```
    Insecure: true
    Location: my-local-registry:5000
    MirrorByDigestOnly: false
    Mirrors: null
    Prefix: my-local-registry:5000
    PullFromMirror: ""
  search:
  - registry.access.redhat.com
  - docker.io
  - my-company-registry.com
```

19.2. 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 for these nodes.
- The user and group **webapp** exist, and must be listed in the **/etc/subuid** and **/etc/subgid** files on the host.
- The user named **dbuser** and a group named **dbgroup** must be already created.

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/rhel-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
```

19.3. 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 **ubi10-httpd** running an HTTP server container from the **registry.access.redhat.com/ubi10/httpd-24** image. The container's web content is mounted from a persistent volume named **ubi10-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 for these nodes.

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: ubi10-httpd
    spec:
      containers:
        - name: ubi10-httpd
          image: registry.access.redhat.com/ubi10/httpd-24
          ports:
            - containerPort: 8080
              hostPort: 8080
          volumeMounts:
            - mountPath: /var/www/html:Z
              name: ubi10-html
      volumes:
        - name: ubi10-html
          persistentVolumeClaim:
            claimName: ubi10-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 **ubi10-httpd** container is based on the **registry.access.redhat.com/ubi10/httpd-24** container image.
- The **ubi10-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 **ubi10-httpd** container can access the content.
- The pod mounts the existing persistent volume named **ubi10-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
```

19.4. 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 for these nodes.
- 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**.
  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.

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

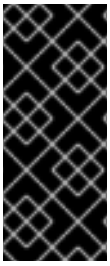
- Ansible vault

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 Postfix 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

You can use the **postfix** RHEL system role to automate configuring Postfix as a null client for 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.

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 for these nodes.

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: 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: "{{ postfix_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: "{{ postfix_default_database_type }}:/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. The postfix_default_database_type variable contains the database type which is set in the "default_database_type" Postfix parameter. On RHEL 10, you have to use **relay_domains: "{{ postfix_default_database_type }}:/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
```

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

You can configure PostgreSQL with TLS encryption using the **postgresql** RHEL system role to automate secure database setup with existing certificates and private keys.

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 for these nodes.
- 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 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)
```

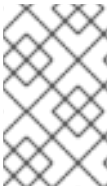
If the output displays a TLS protocol version and cipher details, the connection works and TLS encryption is enabled.

Additional resources

- Ansible vault

21.2. CONFIGURING POSTGRESQL WITH A TLS CERTIFICATE ISSUED FROM IDM BY USING THE POSTGRESQL RHEL SYSTEM ROLE

You can configure PostgreSQL with TLS encryption using the **postgresql** RHEL system role to automate secure database setup with certificates issued from Identity Management (IdM) in Red Hat Enterprise Linux and managed by the **certmonger** service.



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 for these nodes.
- 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 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

- 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 Red Hat Lightspeed-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 Lightspeed. Additionally, with **rhc**, you can:

- Configure connections to Red Hat Lightspeed
- Enable and disable repositories
- Configure the proxy to use for the connection
- Configure Red Hat Lightspeed remediations and, auto updates
- Set the release of the system
- Configure Red Hat Lightspeed 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 Lightspeed when you register it.

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 for these nodes.

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

- 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

- Ansible Vault
- Registering a host to Satellite

22.2. DISABLING THE CONNECTION TO RED HAT LIGHTSPEED 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 Lightspeed. You can disable Red Hat Lightspeed by using the **rhc** RHEL system role, if not required.

Red Hat Lightspeed is a managed service in the Hybrid Cloud Console that uses predictive analytics, remediation capabilities, and deep domain expertise to simplify complex operational 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 for these 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: 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 Lightspeed for proactive analytics and recommendations.
For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.rhc/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
```

22.3. 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 for these nodes.
- 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**.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.rhc/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
```

22.4. LOCKING THE SYSTEM TO A PARTICULAR RELEASE BY USING THE RHC RHEL SYSTEM ROLE

You can lock your system to a specific RHEL release to maintain stability and prevent unintended updates in production environments.

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 for these nodes.
- 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.
For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.rhc/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

- Red Hat Enterprise Linux (RHEL) Extended Update Support (EUS) Overview (Red Hat Knowledgebase)

22.5. 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 for these nodes.

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.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.rhc/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

- Ansible Vault

22.6. MANAGING AUTO UPDATES OF RED HAT LIGHTSPEED RULES BY USING THE RHC RHEL SYSTEM ROLE

You can enable or disable the automatic collection rule updates for Red Hat Lightspeed by using the **rhc** RHEL system role. By default, when you connect your system to Red Hat Lightspeed, 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 for these nodes.
- 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 Lightspeed 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 Lightspeed.
For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.rhc/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

- Ansible Vault

22.7. CONFIGURING RED HAT LIGHTSPEED REMEDIATIONS BY USING THE RHC RHEL SYSTEM ROLE

You can use the **rhc** RHEL system role to configure Red Hat Lightspeed remediations on your systems. When you connect your system to Red Hat Lightspeed, it is enabled by default.

You can use **rhc** to ensure your system is ready for remediation when connected directly to Red Hat. For more information about Red Hat Lightspeed remediations, see Red Hat Lightspeed Remediations .

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 for these nodes.
- You have Red Hat Lightspeed 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
```

22.8. CONFIGURING RED HAT LIGHTSPEED TAGS BY USING THE RHC RHEL SYSTEM ROLE

You can use the **rhc** RHEL system role to configure Red Hat Lightspeed tags. With these tags you can efficiently filter and group systems based on attributes, such as their location. This 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 for these nodes.

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

- Ansible Vault
- Custom system tagging

22.9. 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 for these nodes.
- 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. For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.rhc/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
```

CHAPTER 23. CONFIGURING SELINUX BY USING RHEL SYSTEM ROLES

You can remotely configure and manage SELinux permissions by using the **selinux** RHEL system role.

For example, use the **selinux** role for the following tasks:

- 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.

23.1. RESTORING THE SELINUX CONTEXT ON DIRECTORIES BY USING THE SELINUX RHEL SYSTEM ROLE

To remotely reset the SELinux context on directories, you can use the **selinux** RHEL system role. With an incorrect SELinux context, applications can fail to access the 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 for these nodes.

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 rhel10.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/
```

23.2. 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 to prevent SELinux denying permission to the service. 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 for these nodes.

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
```

23.3. 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 for these nodes.
- 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.

CHAPTER 24. 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 by using an SSH client.
- Secure file transfers: the Secure File Transfer Protocol (SFTP) provided by OpenSSH enables 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.

24.1. 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 ::
```

24.2. CONFIGURING OPENSSH SERVERS BY USING THE SSHD RHEL SYSTEM ROLE

You can use the **sshd** RHEL system role to configure multiple OpenSSH servers for secure remote access.

The role ensures 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 in Red Hat Enterprise Linux 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 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 for these nodes.

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_config:
          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 to login by using a 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.d/00-ansible_system_role.conf
#
# Ansible managed
#
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>
```

24.3. 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 with a different playbook. You can use the non-exclusive configuration to apply changes only to selected configuration options.

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 for these nodes.

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_config:
          # 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_config:
          # 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 a 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
```

24.4. 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 by using the **sshd** RHEL system role.

Override the system-wide cryptographic policy 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, SHA-1 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 for these nodes.

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.

NOTE

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
...
```

24.5. HOW THE SSH RHEL SYSTEM ROLE MAPS SETTINGS FROM A PLAYBOOK TO THE CONFIGURATION FILE

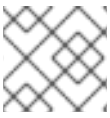
In the **ssh** RHEL system role playbook, you can define the parameters for the client SSH configuration file. If you do not specify these settings, the role produces a global **ssh_config** file that matches the RHEL defaults.

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.

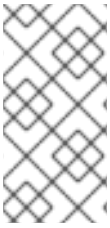
24.6. CONFIGURING OPENSSH CLIENTS BY USING THE SSH RHEL SYSTEM ROLE

You can use the **ssh** RHEL system role to configure multiple OpenSSH clients.

OpenSSH clients 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 in Red Hat Enterprise 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 for these nodes.

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
```

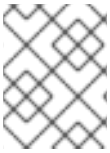
CHAPTER 25. MANAGING LOCAL STORAGE BY USING RHEL SYSTEM ROLES

To manage Logical Volume Manager (LVM) and local file systems (FS) by using Ansible, you can use the **storage** role.

Using the **storage** role enables you to automate administration of file systems on disks and logical volumes on multiple machines.

25.1. CREATING AN XFS FILE SYSTEM ON A BLOCK DEVICE BY USING THE STORAGE RHEL SYSTEM ROLE

You can use the **storage** RHEL system role to automate the creation of an XFS file system on block devices.



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 for these nodes.

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 settings 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.

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
```

25.2. PERSISTENTLY MOUNTING A FILE SYSTEM BY USING THE STORAGE RHEL SYSTEM ROLE

You can use the **storage** RHEL system role to persistently mount file systems to ensure they remain available across system reboots and are automatically mounted on startup. If the file system on the device you specified in the playbook does not exist, the role creates it.

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 for these nodes.

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_safe_mode: false

        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
```

25.3. CREATING OR RESIZING A LOGICAL VOLUME BY USING THE STORAGE RHEL SYSTEM ROLE

You can use the **storage** RHEL system role to create and resize Logical Volume Manager (LVM) logical volumes. The role automatically creates volume groups if they do not exist.

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 for these nodes.

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_safe_mode: false

        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'
```

25.4. 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 for these nodes.

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:
              - /dev/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'
```

25.5. CREATING AND MOUNTING A FILE SYSTEM BY USING THE STORAGE RHEL SYSTEM ROLE

You can use the **storage** RHEL system role to create and mount file systems that persist across reboots. The role automatically adds entries to **/etc/fstab** to ensure persistent mounting.

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 for these nodes.

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_safe_mode: false

      storage_volumes:
        - name: barefs
          type: disk
          disks:
            - sdb
          fs_type: ext4
          fs_label: label-name
          mount_point: /mnt/data
```

   The settings 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
```

25.6. CONFIGURING A RAID VOLUME BY USING THE STORAGE 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 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 for these nodes.

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'
```

25.7. CONFIGURING AN LVM VOLUME GROUP ON RAID BY USING THE STORAGE RHEL SYSTEM ROLE

You can use the **storage** RHEL system role to configure LVM volume groups on RAID arrays.

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 for these nodes.

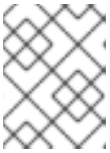
Procedure

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.



NOTE

Setting **raid_level** at the **storage_pool** level creates an MD RAID array first, and then builds an LVM volume group on top of it.

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

- Managing RAID

25.8. CONFIGURING A STRIPE SIZE FOR RAID LVM VOLUMES BY USING THE STORAGE RHEL SYSTEM ROLE

You can use the **storage** RHEL system role to configure stripe sizes for RAID LVM volumes.

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 for these nodes.

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.

   NOTE

   Setting **raid_level** at the **volumes** level creates LVM RAID logical volumes.

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:

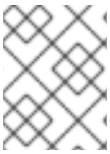
```
# ansible managed-node-01.example.com -m command -a 'lvs -o+stripesize
/dev/my_pool/my_volume'
```

Additional resources

- Managing RAID

25.9. 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 for these nodes.

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
```

25.10. 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 for these 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:

```
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 }}"
            encryption_cipher: <cipher>
            encryption_key_size: <key_size>
            encryption_luks_version: luks2
```

The settings specified in the example playbook include the following:

encryption_cipher: *<cipher>*

Specifies the LUKS cipher. Possible values are: **twofish-xts-plain64**, **serpent-xts-plain64**, and **aes-xts-plain64** (default).

encryption_key_size: *<key_size>*

Specifies the LUKS key size. The default is **512** bit.

encryption_luks_version: luks2

Specifies the LUKS version. The default is **luks2**.

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

- 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: bdf6463f-6b3f-4e55-a0a6-1a66f0152a46
Label: (no label)
Subsystem: (no subsystem)
Flags: (no flags)

Data segments:
0: crypt
offset: 16777216 [bytes]
length: (whole device)
cipher: aes-cbc-essiv:sha256
sector: 512 [bytes]

Keyslots:
0: luks2
Key: 256 bits
Priority: normal
Cipher: aes-cbc-essiv:sha256
Cipher key: 256 bits
```

Additional resources

- Encrypting block devices by using LUKS
- Ansible vault

25.11. 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 for these nodes.
- **lvmlockd** is configured on the managed node. For more information, see Configuring LVM to share SAN disks among multiple machines.
- You have a working cluster environment with shared storage and the storage RHEL system role enabled on each node.

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
                fs_type: gfs2
        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
```

25.12. RESIZING PHYSICAL VOLUMES BY USING THE STORAGE RHEL SYSTEM ROLE

With the **storage** system role, you can resize Logical Volume Manager (LVM) physical volumes after resizing the underlying storage or disks from outside of the host. For example, you increased the size of a virtual disk and want to use the extra space in an existing LVM.

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 for these nodes.
- The size of the underlying block storage has been changed.

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: Resize LVM PV size
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.storage
      vars:
        storage_pools:
          - name: myvg
            disks: ["sdf"]
            type: lvm
            grow_to_fill: true
```

The settings specified in the example playbook include the following:

grow_to_fill

true The role automatically expands the storage volume to use any new capacity on the disk.
false The role leaves the storage volume at its current size, even if the underlying disk has grown.

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

1. Verify the **grow_to_fill** setting works as expected. Prepare a test PV and VG:

```
# pvcreate /dev/sdf
# vgcreate myvg /dev/sdf
```

2. Check and record the initial physical volume size:

```
# pvs
```

3. Edit the playbook to set **grow_to_fill: false** and run the playbook.
4. Check the volume size and verify that it remained unchanged.
5. Edit the playbook to set **grow_to_fill: true** and re-run the playbook.
6. Check the volume size and verify that it has expanded.

CHAPTER 26. USING THE SUDO RHEL SYSTEM ROLE

You can consistently configure the **/etc/sudoers** files on multiple systems by using the **sudo** RHEL system role.

26.1. APPLYING CUSTOM SUDOERS CONFIGURATION BY USING RHEL SYSTEM ROLES

You can use the **sudo** RHEL system role to apply custom **sudoers** configuration on your managed nodes. That way, you can define which users can run which commands on which hosts, with better configuration efficiency and more granular control.

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 for these nodes.

Procedure

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

```
---
- name: "Configure sudo"
  hosts: managed-node-01.example.com
  tasks:
    - name: "Apply custom /etc/sudoers configuration"
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.sudo
      vars:
        sudo_sudoers_files:
          - path: "/etc/sudoers"
            user_specifications:
              - users:
                  - <user_name>
                hosts:
                  - <host_name>
                commands:
                  - <path_to_command_binary>
```

The settings specified in the playbook include the following:

users

The list of users that the rule applies to.

hosts

The list of hosts that the rule applies to. You can use **ALL** for all hosts.

commands

The list of commands that the rule applies to. You can use **ALL** for all commands.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.sudo/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, verify that the playbook applied the new rules.

```
# cat /etc/sudoers | tail -n1
<user_name> <host_name>= <path_to_command_binary>
```

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 **systemd** 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 for these nodes.

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
```

27.2. DEPLOYING SYSTEMD DROP-IN FILES BY USING THE SYSTEMD RHEL SYSTEM ROLE

Systemd applies drop-in files on top of settings 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 for these nodes.

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 activated 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
```

27.3. DEPLOYING SYSTEMD 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 for these nodes.

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
...
```

27.4. DEPLOYING SYSTEMD USER UNITS BY USING THE SYSTEMD RHEL SYSTEM ROLE

You can create per-user unit files for custom applications, and systemd reads them from the **/home/*<username>*/.config/systemd/user/** directory. By using the **systemd** RHEL system role, you can automate the deployment of custom unit files for individual users.

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 for these nodes.
- The user you specify in the playbook for the systemd unit exists.

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
RemainAfterExit=yes

[Install]
WantedBy=multi-user.target
```

   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 for a user
```

```
    ansible.builtin.include_role:
      name: rhel-system-roles.systemd
    vars:
      systemd_unit_file_templates:
        - item: example.service.j2
          user: <username>
      systemd_enabled_units:
        - item: example.service
          user: <username>
      systemd_started_units:
        - item: example.service
          user: <username>
```

IMPORTANT

The **systemd** RHEL system role does not create new users, and it returns an error if you specify a non-existent user in the playbook.

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 --user -M
<username>@ status example.service'
...
● example.service - Example systemd service unit file
     Loaded: loaded (/home/<username>/.config/systemd/user/example.service; enabled;
preset: disabled)
     Active: active (exited) since Wed 2025-03-05 13:33:36 CET; 45s ago
...
```

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. By using the **timesync** RHEL system role, you can automate the configuration of time synchronization on RHEL.

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. By using the **timesync** RHEL system role, you can automate the configuration of RHEL 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 for these nodes.

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

- 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

By using the Network Time Security (NTS) mechanism, clients establish a TLS-encrypted connection to the server and authenticate Network Time Protocol (NTP) packets. By using the **timesync** RHEL system role, you can automate the configuration of RHEL NTP clients with 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 for these nodes.
- 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 number of 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

- 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 for these nodes.

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:

```
# 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>
```

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 for these nodes.

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 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.
For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.tlog/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. 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.

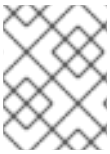
   Note that the value of the **_COMM** field is shortened due to a 15 character limit.

   b. Play back a session:

```
# tlog-play -r journal -M TLOG_REC=<recording_id>
```

CHAPTER 30. CONFIGURING IPSEC VPN CONNECTIONS BY USING RHEL SYSTEM ROLES

Configure IPsec VPN connections to establish encrypted tunnels over untrusted networks and ensure the integrity of data in transit. By using the RHEL system roles, you can automate the setup for use cases, such as connecting branch offices to headquarters.



NOTE

The **vpn** RHEL system role can only create VPN configurations that use pre-shared keys (PSKs) or certificates to authenticate peers to each other.

30.1. CONFIGURING AN IPSEC HOST-TO-HOST VPN WITH PSK AUTHENTICATION BY USING THE VPN RHEL SYSTEM ROLE

A host-to-host VPN establishes an encrypted connection between two devices, allowing applications to communicate safely over an insecure network. By using the **vpn** RHEL system role, you can automate the process of creating IPsec host-to-host connections.

For authentication, a pre-shared key (PSK) is a straightforward method that uses a single, shared secret known only to the two peers. This approach is simple to configure and ideal for basic setups where ease of deployment is a priority. However, you must keep the key strictly confidential. An attacker with access to the key can compromise the connection.

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 for these nodes.

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 peers 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 ***<peer_A>*-to-*<peer_B>***, for example, **managed-node-01.example.com-to-managed-node-02.example.com**. Note that the role cannot configure Libreswan on external (unmanaged) nodes. You must manually create the configuration on these peers.

auth_method: psk

Enables PSK authentication between the peers. The role uses **openssl** on the control node to create the PSK.

auto: *<startup_method>*

Specifies the startup 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.

30.2. CONFIGURING AN IPSEC HOST-TO-HOST VPN WITH PSK AUTHENTICATION AND SEPARATE DATA AND CONTROL PLANES BY USING THE VPN RHEL SYSTEM ROLE

Use the **vpn** RHEL system role to automate the process of creating an IPsec host-to-host VPN. To enhance security by minimizing the risk of control messages being intercepted or disrupted, configure separate connections for both the data traffic and the control traffic.

A host-to-host VPN establishes a direct, secure, and encrypted connection between two devices, allowing applications to communicate safely over an insecure network, such as the internet.

For authentication, a pre-shared key (PSK) is a straightforward method that uses a single, shared secret known only to the two peers. This approach is simple to configure and ideal for basic setups where ease of deployment is a priority. However, you must keep the key strictly confidential. An attacker with access to the key can compromise the connection.

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 for these nodes.

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 cannot 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: *<startup_method>*

Specifies the startup 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.

30.3. CONFIGURING AN IPSEC SITE-TO-SITE VPN WITH PSK AUTHENTICATION BY USING THE VPN RHEL SYSTEM ROLE

A site-to-site VPN establishes an encrypted tunnel between two distinct networks, seamlessly linking them across an insecure public network. By using the **vpn** RHEL system role, you can automate the process of creating IPsec site-to-site VPN connections.

A site-to-site VPN enables devices in a branch office to access resources at a corporate headquarters just as if they were all part of the same local network.

For authentication, a pre-shared key (PSK) is a straightforward method that uses a single, shared secret known only to the two peers. This approach is simple to configure and ideal for basic setups where ease of deployment is a priority. However, you must keep the key strictly confidential. An attacker with access to the key can compromise the connection.

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 for these nodes.

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:
                subnets:
                  - 192.0.2.0/24
              managed-node-02.example.com:
                subnets:
                  - 198.51.100.0/24
                  - 203.0.113.0/24
```

```
    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 gateways 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 ***<gateway_A>*-to-*<gateway_B>***, for example, **managed-node-01.example.com-to-managed-node-02.example.com**. Note that the role cannot configure Libreswan on external (unmanaged) nodes. You must manually create the configuration on these peers.

subnets: *<yaml_list_of_subnets>*

Defines subnets in classless inter-domain routing (CIDR) format that are connected through the tunnel.

auth_method: psk

Enables PSK authentication between the peers. The role uses **openssl** on the control node to create the PSK.

auto: *<startup_method>*

Specifies the startup 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.

30.4. CONFIGURING AN IPSEC MESH VPN WITH CERTIFICATE-BASED AUTHENTICATION BY USING THE VPN RHEL SYSTEM ROLE

An IPsec mesh creates a fully interconnected network where every server can communicate securely and directly with every other server. By using the **vpn** RHEL system role, you can automate configuring a VPN mesh with certificate-based authentication among managed nodes.

An IPsec mesh is ideal for distributed database clusters or high-availability environments that span multiple data centers or cloud providers. Establishing a direct, encrypted tunnel between each pair of servers ensures secure communication without a central bottleneck.

For authentication, using digital certificates managed by a Certificate Authority (CA) offers a highly secure and scalable solution. Each host in the mesh presents a certificate signed by a trusted CA. This method provides strong, verifiable authentication and simplifies user management. Access can be granted or revoked centrally at the CA, and Libreswan enforces this by checking each certificate against a certificate revocation list (CRL), denying access if a certificate appears on the list.

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 for these nodes.
- You prepared a PKCS #12 file for each managed node:
  - Each file contains:
    - The private key of the server
    - The server certificate
    - The CA certificate
    - If required, intermediate certificates

- The files are named ***<managed_node_name_as_in_the_inventory>*.p12**.
- The files are stored in the same directory as the playbook.
- The server certificate contains the following fields:
  - Extended Key Usage (EKU) is set to **TLS Web Server Authentication**.
  - Common Name (CN) or Subject Alternative Name (SAN) is set to the fully-qualified domain name (FQDN) of the host.
  - X509v3 CRL distribution points contain URLs to Certificate Revocation Lists (CRLs).

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: Initialize IPsec NSS database if not initialized
    ansible.builtin.command:
      cmd: ipsec initnss
    when: db_files.matched == 0

  - 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 can be encrypted and which of them should continue using plain text connections.

auth_method: cert

Enables certificate-based authentication. This requires that you specify 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 connection 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'
```