Red Hat Enterprise Linux 8

Administration and configuration tasks using System Roles in RHEL

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

Last Updated: 2021-01-06

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

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

Legal Notice

Abstract

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

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE 4

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION 5

CHAPTER 1. GETTING STARTED WITH RHEL SYSTEM ROLES 6
1.1. INTRODUCTION TO RHEL SYSTEM ROLES 6
1.2. RHEL SYSTEM ROLES TERMINOLOGY 6
1.3. APPLYING A ROLE 7
1.4. ADDITIONAL RESOURCES 9

CHAPTER 2. INSTALLING RHEL SYSTEM ROLES 10
2.1. INSTALLING RHEL SYSTEM ROLES IN YOUR SYSTEM 10

CHAPTER 3. USING ANSIBLE ROLES TO PERMANENTLY CONFIGURE KERNEL PARAMETERS 11
3.1. INTRODUCTION TO THE KERNEL SETTINGS ROLE 11
3.2. APPLYING SELECTED KERNEL PARAMETERS USING THE KERNEL SETTINGS ROLE 12

CHAPTER 4. USING SYSTEM ROLES TO CONFIGURE NETWORK CONNECTIONS 15
4.1. CONFIGURING AN ETHERNET CONNECTION 15
4.1.1. Configuring a static Ethernet connection using RHEL System Roles 15
4.1.2. Configuring a dynamic Ethernet connection using RHEL System Roles 16
4.2. CONFIGURING VLAN TAGGING 18
4.2.1. Configuring VLAN tagging using System Roles 18
4.3. CONFIGURING A NETWORK BRIDGE 20
4.3.1. Configuring a network bridge using RHEL System Roles 20
4.4. CONFIGURING NETWORK BONDING 22
4.4.1. Configuring a network bond using RHEL System Roles 22
4.5. AUTHENTICATING A RHEL CLIENT TO THE NETWORK USING THE 802.1X STANDARD 24
4.5.1. Configuring a static Ethernet connection with 802.1X network authentication using RHEL System Roles 24
4.6. MANAGING THE DEFAULT GATEWAY SETTING 26
4.6.1. Setting the default gateway on an existing connection using System Roles 26
4.7. CONFIGURING STATIC ROUTES 28
4.7.1. Configuring a static route using RHEL System Roles 28
4.8. CONFIGURING ETHTOOL OFFLOAD FEATURES 31
4.8.1. Using System Roles to set ethtool features 31

CHAPTER 5. CONFIGURING SELINUX USING SYSTEM ROLES 34
5.1. INTRODUCTION TO THE SELINUX SYSTEM ROLE 34
5.2. USING THE SELINUX SYSTEM ROLE TO APPLY SELINUX SETTINGS ON MULTIPLE SYSTEMS 35

CHAPTER 6. USING THE LOGGING SYSTEM ROLE 36
6.1. THE LOGGING SYSTEM ROLE 36
6.2. LOGGING SYSTEM ROLE PARAMETERS 36
6.3. APPLYING A LOCAL LOGGING SYSTEM ROLE 37
6.4. APPLYING A REMOTE LOGGING SOLUTION USING THE LOGGING SYSTEM ROLE 39
6.5. ADDITIONAL RESOURCES 42

CHAPTER 7. USING THE CLEVIS AND TANG SYSTEM ROLES 43
7.1. INTRODUCTION TO THE CLEVIS AND TANG SYSTEM ROLES 43
7.2. USING THE NBDE_SERVER SYSTEM ROLE FOR SETTING UP MULTIPLE TANG SERVERS 43
7.3. USING THE NBDE_CLIENT SYSTEM ROLE FOR SETTING UP MULTIPLE CLEVIS CLIENTS 44

CHAPTER 8. REQUESTING CERTIFICATES USING RHEL SYSTEM ROLES 47

8.1. THE CERTIFICATE SYSTEM ROLE 47
8.2. REQUESTING A NEW SELF-SIGNED CERTIFICATE USING THE CERTIFICATE SYSTEM ROLE 47
8.3. REQUESTING A NEW CERTIFICATE FROM IDM CA USING THE CERTIFICATE SYSTEM ROLE 49
8.4. SPECIFYING COMMANDS TO RUN BEFORE OR AFTER CERTIFICATE ISSUANCE USING THE CERTIFICATE SYSTEM ROLE 50

CHAPTER 9. CONFIGURING KDUMP USING RHEL SYSTEM ROLES 53
9.1. THE KDUMP RHEL SYSTEM ROLE 53
9.2. KDUMP ROLE PARAMETERS 53
9.3. CONFIGURING KDUMP USING RHEL SYSTEM ROLES 53

CHAPTER 10. CONFIGURING STORAGE USING RHEL SYSTEM ROLES 55
10.1. INTRODUCTION TO THE STORAGE ROLE 55
10.2. PARAMETERS THAT IDENTIFY A STORAGE DEVICE IN THE STORAGE SYSTEM ROLE 55
10.3. CREATING AN XFS FILE SYSTEM ON A BLOCK DEVICE USING RHEL SYSTEM ROLES 56
10.3.1. Example Ansible playbook to create an XFS file system on a block device 56
10.4. PERSISTENTLY MOUNTING A FILE SYSTEM USING RHEL SYSTEM ROLES 57
10.4.1. Example Ansible playbook to persistently mount a file system 57
10.5. ENABLING ONLINE BLOCK DISCARD USING RHEL SYSTEM ROLES 58
10.5.1. Example Ansible playbook to enable online block discard 58
10.6. CREATING AND MOUNTING EXT3 FILE SYSTEMS USING RHEL SYSTEM ROLES 58
10.6.1. Example Ansible playbook to create and mount an ext3 file system 59
10.7. CREATING AND MOUNTING EXT4 FILE SYSTEMS USING RHEL SYSTEM ROLES 59
10.7.1. Example Ansible playbook to create and mount an Ext4 file system 59
10.8. MANAGING LVM LOGICAL VOLUMES USING RHEL SYSTEM ROLES 60
10.8.1. Example Ansible playbook to manage logical volumes 60
10.8.2. Additional resources 61
10.9. CONFIGURING A RAID VOLUME USING THE STORAGE SYSTEM ROLE 61
10.10. CONFIGURING AN LVM POOL WITH RAID USING THE STORAGE SYSTEM ROLE 62
10.11. MANAGING VOLUMES ENCRYPTED WITH LUKS USING RHEL SYSTEM ROLES 64
10.11.1. Creating a LUKS encrypted volume using the storage role 64

CHAPTER 11. CONFIGURING TIME SYNCHRONIZATION USING RHEL SYSTEM ROLES 66
11.1. THE TIMESYNC SYSTEM ROLE 66
11.2. APPLYING THE TIMESYNC SYSTEM ROLE FOR A SINGLE POOL OF SERVERS 66
11.3. TIMESYNC SYSTEM ROLES VARIABLES 67

CHAPTER 12. MONITORING PERFORMANCE USING RHEL SYSTEM ROLES 68
12.1. INTRODUCTION TO THE METRICS SYSTEM ROLE 68
12.2. USING THE METRICS SYSTEM ROLE TO MONITOR YOUR LOCAL SYSTEM WITH VISUALIZATION 69
12.3. USING THE METRICS SYSTEM ROLE TO SETUP A FLEET OF INDIVIDUAL SYSTEMS TO MONITOR THEMSELVES 69
12.4. USING THE METRICS SYSTEM ROLE TO MONITOR A FLEET OF MACHINES CENTRALLY VIA YOUR LOCAL MACHINE 70

CHAPTER 13. CONFIGURING A SYSTEM FOR SESSION RECORDING USING THE TLOG RHEL SYSTEM ROLES 72
13.1. THE TLOG SYSTEM ROLE 72
13.2. COMPONENTS AND PARAMETERS OF THE TLOG SYSTEM ROLES 72
13.3. DEPLOYING THE TLOG RHEL SYSTEM ROLE 72
13.4. RECORDING A SESSION USING THE DEPLOYED TLOG SYSTEM ROLE IN THE CLI 74
13.5. WATCHING A RECORDED SESSION USING THE CLI 75

MAKING OPEN SOURCE MORE INCLUSIVE

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

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Please let us know how we could make it better. To do so:

- For simple comments on specific passages:
  1. Make sure you are viewing the documentation in the *Multi-page HTML* format. In addition, ensure you see the **Feedback** button in the upper right corner of the document.
  2. Use your mouse cursor to highlight the part of text that you want to comment on.
  3. Click the **Add Feedback** pop-up that appears below the highlighted text.
  4. Follow the displayed instructions.
- For submitting more complex feedback, create a Bugzilla ticket:
  1. Go to the Bugzilla website.
  2. As the Component, use **Documentation**.
  3. Fill in the **Description** field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.
  4. Click **Submit Bug**.

CHAPTER 1. GETTING STARTED WITH RHEL SYSTEM ROLES

This section explains what RHEL System Roles are. Additionally, it describes how to apply a particular role through an Ansible playbook to perform various system administration tasks.

1.1. INTRODUCTION TO RHEL SYSTEM ROLES

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

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

- kdump
- network
- selinux
- storage
- certificate
- kernel_settings
- logging
- metrics
- nbde_client and nbde_server
- timesync
- tlog

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

Additional resources

- For RHEL System Roles overview, see the Red Hat Enterprise Linux (RHEL) System Roles Red Hat Knowledgebase article.
- For information on a particular role, see the documentation under the **/usr/share/doc/rhel-system-roles** directory. This documentation is installed automatically with the **rhel-system-roles** package.
- Introduction to the SELinux system role
- Introduction to the storage role

1.2. RHEL SYSTEM ROLES TERMINOLOGY

You can find the following terms across this documentation:

System Roles terminology

Ansible playbook

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

Control node

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

Inventory

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

Managed nodes

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

1.3. APPLYING A ROLE

The following procedure describes how to apply a particular role.

Prerequisites

- The **rhel-system-roles** package is installed on the system that you want to use as a control node:

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

- The Ansible Engine repository is enabled, and the **ansible** package is installed on the system that you want to use as a control node. You need the **ansible** package to run playbooks that use RHEL System Roles.
  - If you do not have a Red Hat Ansible Engine Subscription, you can use a limited supported version of Red Hat Ansible Engine provided with your Red Hat Enterprise Linux subscription. In this case, follow these steps:
    1. Enable the RHEL Ansible Engine repository:

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

    2. Install Ansible Engine:

```
# yum install ansible
```

  - If you have a Red Hat Ansible Engine Subscription, follow the procedure described in How do I Download and Install Red Hat Ansible Engine?.

- You are able to create an Ansible playbook.
  Playbooks represent Ansible's configuration, deployment, and orchestration language. By using playbooks, you can declare and manage configurations of remote machines, deploy multiple remote machines or orchestrate steps of any manual ordered process.

A playbook is a list of one or more **plays**. Every **play** can include Ansible variables, tasks, or roles.

Playbooks are human-readable, and they are expressed in the **YAML** format.

For more information about playbooks, see Ansible documentation.

Procedure

1. Create an Ansible playbook including the required role.
   The following example shows how to use roles through the **roles:** option for a given **play**:

```
---
- hosts: webservers
  roles:
    - rhel-system-roles.network
    - rhel-system-roles.timesync
```

   For more information on using roles in playbooks, see Ansible documentation.

   See Ansible examples for example playbooks.

NOTE

Every role includes a README file, which documents how to use the role and supported parameter values. You can also find an example playbook for a particular role under the documentation directory of the role. Such documentation directory is provided by default with the **rhel-system-roles** package, and can be found in the following location:

```
/usr/share/doc/rhel-system-roles/SUBSYSTEM/
```

Replace *SUBSYSTEM* with the name of the required role, such as **selinux**, **kdump**, **network**, **timesync**, or **storage**.

2. Verify the playbook syntax:

```
# ansible-playbook --syntax-check name.of.the.playbook
```

   The **ansible-playbook** command offers a **--syntax-check** option that you can use to verify the syntax of a playbook.

3. Execute the playbook on targeted hosts by running the **ansible-playbook** command:

```
# ansible-playbook -i name.of.the.inventory name.of.the.playbook
```

   An inventory is a list of systems against which Ansible works. For more information on how to create and inventory, and how to work with it, see Ansible documentation.

   If you do not have an inventory, you can create it at the time of running **ansible-playbook**:

   If you have only one targeted host against which you want to run the playbook, use:

```
# ansible-playbook -i host1, name.of.the.playbook
```

If you have multiple targeted hosts against which you want to run the playbook, use:

```
# ansible-playbook -i host1,host2,....,hostn name.of.the.playbook
```

Additional resources

- For more detailed information on using the **ansible-playbook** command, see the **ansible-playbook** man page.

1.4. ADDITIONAL RESOURCES

- For RHEL System Roles overview, see the Red Hat Enterprise Linux (RHEL) System Roles Red Hat Knowledgebase article.
- Managing local storage using RHEL System Roles
- Deploying the same SELinux configuration on multiple systems using RHEL System Roles

CHAPTER 2. INSTALLING RHEL SYSTEM ROLES

Before starting to use System Roles, you must install it in your system.

2.1. INSTALLING RHEL SYSTEM ROLES IN YOUR SYSTEM

This paragraph is the procedure module introduction: a short description of the procedure.

Prerequisites

- You have a Red Hat Ansible Engine Subscription. See the procedure How do I Download and Install Red Hat Ansible Engine?
- You have Ansible packages installed in the system you want to use as a control node:

Procedure

1. Install the **rhel-system-roles** package on the system that you want to use as a control node:

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

   If you do not have a Red Hat Ansible Engine Subscription, you can use a limited supported version of Red Hat Ansible Engine provided with your Red Hat Enterprise Linux subscription. In this case, follow these steps:

   a. Enable the RHEL Ansible Engine repository:

```
# subscription-manager refresh

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

   b. Install Ansible Engine:

```
# yum install ansible
```

As a result, you are able to create an Ansible playbook.

Additional resources

- For RHEL System Roles overview, see the Red Hat Enterprise Linux (RHEL) System Roles
- For more detailed information on using the ansible-playbook command, see the ansible-playbook man page.

CHAPTER 3. USING ANSIBLE ROLES TO PERMANENTLY CONFIGURE KERNEL PARAMETERS

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

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

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

3.1. INTRODUCTION TO THE KERNEL SETTINGS ROLE

RHEL System Roles is a collection of roles and modules from Ansible Automation Platform that provide a consistent configuration interface to remotely manage multiple systems.

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

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

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

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

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

Additional resources

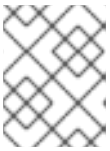
- For a detailed reference on **kernel_settings** role variables and for the example playbooks, install the **rhel-system-roles** package, and see the **README.md** and **README.html** files in the **/usr/share/doc/rhel-system-roles/kernel_settings/** directory.
- For more information about playbooks, see Working with playbooks in Ansible documentation.
- For more information on creating and using inventories, see How to build your inventory in Ansible documentation.

3.2. APPLYING SELECTED KERNEL PARAMETERS USING THE KERNEL SETTINGS ROLE

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

Prerequisites

- Your Red Hat Ansible Engine subscription is attached to the system, also called *control machine*, from which you want to run the **kernel_settings** role. See the How do I download and install Red Hat Ansible Engine article for more information.
- Ansible Engine repository is enabled on the control machine.
- Ansible Engine is installed on the control machine.



NOTE

You do not need to have Ansible Engine installed on the systems, also called *managed hosts*, where you want to configure the kernel parameters.

- The **rhel-system-roles** package is installed on the control machine.
- An inventory of managed hosts is present on the control machine and Ansible Engine is able to connect to them.

Procedure

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

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

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

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

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

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

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

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

   b. Insert the following content into the file:

```
[defaults]
inventory = ./inventory

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

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

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

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

   ```
   # vi /home/jdoe/<ansible_project_name>/kernel_roles.yml
   ```

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

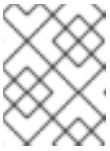
   b. Insert the following content into the file:

   ```
   ---
   - name: Configure kernel settings
     hosts: testingservers

     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

     roles:
       - linux-system-roles.kernel_settings
   ```

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

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

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



NOTE

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

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

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

playbook: kernel-roles.yml
```

This example shows the successful verification of a playbook.

5. Execute your playbook.

```
# ansible-playbook kernel-roles.yml
BECOME password:

PLAY [Configure kernel settings] ... PLAY RECAP **
fdoe@192.168.122.226      : ok=10   changed=4    unreachable=0    failed=0    skipped=6
rescued=0    ignored=0
pdoe@192.168.122.98       : ok=10   changed=4    unreachable=0    failed=0    skipped=6
rescued=0    ignored=0
```

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

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

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

Additional resources

- For more information about RHEL System Roles, see Getting started with RHEL System Roles.
- For more information about all currently supported variables in **kernel_settings**, see **README.html** and **README.md** files in the **/usr/share/doc/rhel-system-roles/kernel_settings/** directory.
- For more details about Ansible inventories, see Working with Inventory in Ansible documentation.
- For more details about Ansible configuration files, see Configuring Ansible in Ansible documentation.
- For more details about Ansible playbooks, see Working With Playbooks in Ansible documentation.
- For more details about Ansible variables, see Using Variables in Ansible documentation.
- For more details about Ansible roles, see Roles in Ansible documentation.

CHAPTER 4. USING SYSTEM ROLES TO CONFIGURE NETWORK CONNECTIONS

The **network** system role on RHEL enables administrators to automate network-related configuration and management tasks using Ansible.

4.1. CONFIGURING AN ETHERNET CONNECTION

This section describes different ways how to configure an Ethernet connection with static and dynamic IP addresses.

4.1.1. Configuring a static Ethernet connection using RHEL System Roles

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

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

Run this procedure on the Ansible control node.

Prerequisites

- The **ansible** and **rhel-system-roles** packages are installed on the control node.
- If you use a different remote user than **root** when you run the playbook, this user has appropriate **sudo** permissions on the managed node.
- The host uses NetworkManager to configure the network.

Procedure

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

   ```
   node.example.com
   ```

2. Create the **~/ethernet-static-IP.yml** playbook with the following content:

   ```
   ---
   - name: Configure an Ethernet connection with static IP
     hosts: node.example.com
     become: true
   ```

```
tasks:
- include_role:
    name: linux-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
        state: up
```

3. Run the playbook:

   - To connect as **root** user to the managed host, enter:

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

   - To connect as a user to the managed host, enter:

     ```
     # ansible-playbook -u user_name --ask-become-pass ~/ethernet-static-IP.yml
     ```

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

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

Additional resources

- For details about the parameters used in **network_connections** and for additional information about the **network** System Role, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file.
- For details about the **ansible-playbook** command, see the **ansible-playbook(1)** man page.

4.1.2. Configuring a dynamic Ethernet connection using RHEL System Roles

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

Prerequisites

- A DHCP server is available in the network.
- The **ansible** and **rhel-system-roles** packages are installed on the control node.
- If you use a different remote user than **root** when you run the playbook, this user has appropriate **sudo** permissions on the managed node.
- The host uses NetworkManager to configure the network.

Procedure

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

```
node.example.com
```

2. Create the **~/ethernet-dynamic-IP.yml** playbook with the following content:

```
---
- name: Configure an Ethernet connection with dynamic IP
  hosts: node.example.com
  become: true
  tasks:
  - include_role:
      name: linux-system-roles.network

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

3. Run the playbook:

   - To connect as **root** user to the managed host, enter:

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

   - To connect as a user to the managed host, enter:

```
# ansible-playbook -u user_name --ask-become-pass ~/ethernet-dynamic-IP.yml
```

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

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

Additional resources

- For details about the parameters used in **network_connections** and for additional information about the **network** System Role, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file.
- For details about the **ansible-playbook** command, see the **ansible-playbook(1)** man page.

4.2. CONFIGURING VLAN TAGGING

This section describes how to configure Virtual Local Area Network (VLAN). A VLAN is a logical network within a physical network. The VLAN interface tags packets with the VLAN ID as they pass through the interface, and removes tags of returning packets.

You create a VLAN interface on top of another interface, such as an Ethernet, bond, team, or bridge device. This interface is called the **parent interface**.

4.2.1. Configuring VLAN tagging using System Roles

You can use the **networking** RHEL System Role to configure VLAN tagging. This procedure describes how to add an Ethernet connection and a VLAN with ID **10** that uses this Ethernet connection. As the parent device, the VLAN connection contains the IP, default gateway, and DNS configurations.

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

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

Prerequisites

- The **ansible** and **rhel-system-roles** packages are installed on the control node.
- If you use a different remote user than **root** when you run the playbook, this user has appropriate **sudo** permissions on the managed node.

Procedure

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

```
node.example.com
```

2. Create the **~/vlan-ethernet.yml** playbook with the following content:

```
---
- name: Configure a VLAN that uses an Ethernet connection
  hosts: node.example.com
  become: true
  tasks:
  - include_role:
      name: linux-system-roles.network

    vars:
```

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

      # Define the VLAN profile
      - name: vlan10
        type: vlan
        ip:
          address:
            - "192.0.2.1/24"
            - "2001:db8:1::1/64"
          gateway4: 192.0.2.254
          gateway6: 2001:db8:1::fffe
          dns:
            - 192.0.2.200
            - 2001:db8:1::ffbb
          dns_search:
            - example.com
        vlan_id: 10
parent: enp1s0
        state: up
```

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

3. Run the playbook:

   - To connect as **root** user to the managed host, enter:

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

   - To connect as a user to the managed host, enter:

     ```
     # ansible-playbook -u user_name --ask-become-pass ~/vlan-ethernet.yml
     ```

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

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

Additional resources

- For details about the parameters used in **network_connections** and for additional information about the **network** System Role, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file.
- For details about the **ansible-playbook** command, see the **ansible-playbook(1)** man page.

4.3. CONFIGURING A NETWORK BRIDGE

A network bridge is a link-layer device which forwards traffic between networks based on a table of MAC addresses. The bridge builds the MAC addresses table by listening to network traffic and thereby learning what hosts are connected to each network. For example, you can use a software bridge on a Red Hat Enterprise Linux 8 host to emulate a hardware bridge or in virtualization environments, to integrate virtual machines (VM) to the same network as the host.

A bridge requires a network device in each network the bridge should connect. When you configure a bridge, the bridge is called **master** and the devices it uses **slave** devices.

You can create bridges on different types of slave devices, such as:

- Physical and virtual Ethernet devices
- Network bonds
- Network teams
- VLAN devices

Due to the IEEE 802.11 standard which specifies the use of 3-address frames in Wi-Fi for the efficient use of airtime, you cannot configure a bridge over Wi-Fi networks operating in Ad-Hoc or Infrastructure modes.

4.3.1. Configuring a network bridge using RHEL System Roles

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

NOTE

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

Prerequisites

- The **ansible** and **rhel-system-roles** packages are installed on the control node.
- If you use a different remote user than **root** when you run the playbook, this user has appropriate **sudo** permissions on the managed node.
- Two or more physical or virtual network devices are installed on the server.

Procedure

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

```
node.example.com
```

2. Create the ~/**bridge-ethernet.yml** playbook with the following content:

```
---
- name: Configure a network bridge that uses two Ethernet ports
```

```
hosts: node.example.com
become: true
tasks:
- include_role:
    name: linux-system-roles.network

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

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

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

3. Run the playbook:

   - To connect as **root** user to the managed host, enter:

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

   - To connect as a user to the managed host, enter:

     ```
     # ansible-playbook -u user_name --ask-become-pass ~/bridge-ethernet.yml
     ```

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

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

Additional resources

- For details about the parameters used in **network_connections** and for additional information about the **network** System Role, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file.
- For details about the **ansible-playbook** command, see the **ansible-playbook(1)** man page.

4.4. CONFIGURING NETWORK BONDING

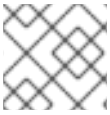
This section describes the basics of network bonding, the differences between bonding and teaming, and how to configure a network bond on Red Hat Enterprise Linux 8.

You can create bonds on different types of slave devices, such as:

- Physical and virtual Ethernet devices
- Network bridges
- Network teams
- VLAN devices

4.4.1. Configuring a network bond using RHEL System Roles

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



NOTE

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

Prerequisites

- The **ansible** and **rhel-system-roles** packages are installed on the control node.
- If you use a different remote user than **root** when you run the playbook, this user has appropriate **sudo** permissions on the managed node.
- Two or more physical or virtual network devices are installed on the server.

Procedure

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

```
node.example.com
```

2. Create the **~/bond-ethernet.yml** playbook with the following content:

```
---
- name: Configure a network bond that uses two Ethernet ports
  hosts: node.example.com
  become: true
  tasks:
  - include_role:
      name: linux-system-roles.network

    vars:
      network_connections:
        # Define the bond profile
        - name: bond0
          type: bond
          interface_name: bond0
          ip:
            address:
              - "192.0.2.1/24"
              - "2001:db8:1::1/64"
            gateway4: 192.0.2.254
            gateway6: 2001:db8:1::fffe
            dns:
              - 192.0.2.200
              - 2001:db8:1::ffbb
            dns_search:
              - example.com
          bond:
            mode: active-backup
          state: up

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

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

3. Run the playbook:

   - To connect as **root** user to the managed host, enter:

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

   - To connect as a user to the managed host, enter:

     ```
     # ansible-playbook -u user_name --ask-become-pass ~/bond-ethernet.yml
     ```

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

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

Additional resources

- For details about the parameters used in **network_connections** and for additional information about the **network** System Role, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file.
- For details about the **ansible-playbook** command, see the **ansible-playbook(1)** man page.

4.5. AUTHENTICATING A RHEL CLIENT TO THE NETWORK USING THE 802.1X STANDARD

Administrators frequently use port-based Network Access Control (NAC) based on the IEEE 802.1X standard to protect a network from unauthorized LAN and Wi-Fi clients. The procedures in this section describe different options to configure network authentication.

4.5.1. Configuring a static Ethernet connection with 802.1X network authentication using RHEL System Roles

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

- A static IPv4 address - **192.0.2.1** with a **/24** subnet mask
- A static IPv6 address - **2001:db8:1::1** with a **/64** subnet mask
- An IPv4 default gateway - **192.0.2.254**
- An IPv6 default gateway - **2001:db8:1::fffe**
- An IPv4 DNS server - **192.0.2.200**
- An IPv6 DNS server - **2001:db8:1::ffbb**
- A DNS search domain - **example.com**
- 802.1X network authentication using the **TLS** Extensible Authentication Protocol (EAP)

Run this procedure on the Ansible control node.

Prerequisites

- The **ansible** and **rhel-system-roles** packages are installed on the control node.
- If you use a different remote user than **root** when you run the playbook, you must have appropriate **sudo** permissions on the managed node.
- The network supports 802.1X network authentication.
- The managed node uses NetworkManager.

- The following files required for TLS authentication exist on the control node:
  - The client key stored in the **/srv/data/client.key** file.
  - The client certificate stored in the **/srv/data/client.crt** file.
  - The Certificate Authority (CA) certificate stored in the **/srv/data/ca.crt** file.

Procedure

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

```
node.example.com
```

2. Create the **~/enable-802.1x.yml** playbook with the following content:

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

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

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

    - include_role:
        name: linux-system-roles.network
      vars:
        network_connections:
          - name: enp1s0
            type: ethernet
            autoconnect: yes
            ip:
              address:
                - 192.0.2.1/24
                - 2001:db8:1::1/64
              gateway4: 192.0.2.254
              gateway6: 2001:db8:1::fffe
              dns:
                - 192.0.2.200
                - 2001:db8:1::ffbb
              dns_search:
                - example.com
```

```
    ieee802_1x:
      identity: user_name
      eap: tls
      private_key: "/etc/pki/tls/private/client.key"
      private_key_password: "password"
      client_cert: "/etc/pki/tls/certs/client.crt"
      ca_cert: "/etc/pki/ca-trust/source/anchors/ca.crt"
      domain_suffix_match: example.com
    state: up
```

3. Run the playbook:

   - To connect as **root** user to the managed host, enter:

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

   - To connect as a user to the managed host, enter:

     ```
     # ansible-playbook -u user_name --ask-become-pass ~/ethernet-static-IP.yml
     ```

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

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

Additional resources

- For details about the parameters used in **network_connections** and for additional information about the **network** System Role, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file.
- For details about the 802.1X parameters, see the **ieee802_1x** section in the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file.
- For details about the **ansible-playbook** command, see the **ansible-playbook(1)** man page.

4.6. MANAGING THE DEFAULT GATEWAY SETTING

The default gateway is a router that forwards network packets when no other route matches the destination of a packet. In a local network, the default gateway is typically the host that is one hop closer to the internet.

4.6.1. Setting the default gateway on an existing connection using System Roles

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

IMPORTANT

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

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

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

Prerequisites

- The **ansible** and **rhel-system-roles** packages are installed on the control node.
- If you use a different remote user than **root** when you run the playbook, this user has appropriate **sudo** permissions on the managed node.

Procedure

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

```
node.example.com
```

2. Create the **~/ethernet-connection.yml** playbook with the following content:

```
---
- name: Configure an Ethernet connection with static IP and default gateway
  hosts: node.example.com
  become: true
  tasks:
  - include_role:
      name: linux-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
```

3. Run the playbook:

   - To connect as **root** user to the managed host, enter:

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

   - To connect as a user to the managed host, enter:

     ```
     # ansible-playbook -u user_name --ask-become-pass ~/ethernet-connection.yml
     ```

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

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

Additional resources

- For details about the parameters used in **network_connections** and for additional information about the **network** System Role, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file.
- For details about the **ansible-playbook** command, see the **ansible-playbook(1)** man page.

4.7. CONFIGURING STATIC ROUTES

By default, and if a default gateway is configured, Red Hat Enterprise Linux forwards traffic for networks that are not directly connected to the host to the default gateway. Using a static route, you can configure that Red Hat Enterprise Linux forwards the traffic for a specific host or network to a different router than the default gateway. This section describes different options how to configure static routes.

4.7.1. Configuring a static route using RHEL System Roles

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

IMPORTANT

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

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

- A static IPv4 address - **198.51.100.20** with a **/24** subnet mask
- A static IPv6 address - **2001:db8:1::1** with a **/64** subnet mask
- An IPv4 default gateway - **198.51.100.254**
- An IPv6 default gateway - **2001:db8:1::fffe**
- An IPv4 DNS server - **198.51.100.200**
- An IPv6 DNS server - **2001:db8:1::ffbb**
- A DNS search domain - **example.com**
- Static routes:
  - **192.0.2.0/24** with gateway **198.51.100.1**
  - **203.0.113.0/24** with gateway **198.51.100.2**

Prerequisites

- The **ansible** and **rhel-system-roles** packages are installed on the control node.
- If you use a different remote user than **root** when you run the playbook, this user has appropriate **sudo** permissions on the managed node.

Procedure

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

```
node.example.com
```

2. Create the **~/add-static-routes.yml** playbook with the following content:

```
---
- name: Configure an Ethernet connection with static IP and additional routes
  hosts: node.example.com
  become: true
  tasks:
  - include_role:
      name: linux-system-roles.network
```

```
    vars:
      network_connections:
        - name: enp7s0
          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
            route:
              - network: 192.0.2.0
                prefix: 24
                gateway: 198.51.100.1
              - network: 203.0.113.0
                prefix: 24
                gateway: 198.51.100.2
          state: up
```

3. Run the playbook:

   - To connect as **root** user to the managed host, enter:

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

   - To connect as a user to the managed host, enter:

     ```
     # ansible-playbook -u user_name --ask-become-pass ~/add-static-routes.yml
     ```

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

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

Verification steps

- Display the routing table:

```
# ip -4 route
default via 198.51.100.254 dev enp7s0 proto static metric 100
192.0.2.0/24 via 198.51.100.1 dev enp7s0 proto static metric 100
203.0.113.0/24 via 198.51.100.2 dev enp7s0 proto static metric 100
...
```

Additional resources

- For details about the parameters used in **network_connections** and for additional information about the **network** System Role, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file.
- For details about the **ansible-playbook** command, see the **ansible-playbook(1)** man page.

4.8. CONFIGURING ETHTOOL OFFLOAD FEATURES

Network interface cards can use the TCP offload engine (TOE) to offload processing certain operations to the network controller to improve the network throughput.

This section describes how to set offload features.

4.8.1. Using System Roles to set ethtool features

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

IMPORTANT

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

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

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

Prerequisites

- The **ansible** and **rhel-system-roles** packages are installed on the control node.

- If you use a different remote user than root when you run the playbook, this user has appropriate **sudo** permissions on the managed node.

Procedure

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

```
node.example.com
```

2. Create the ~/**configure-ethernet-device-with-ethtool-features.yml** playbook with the following content:

```
---
- name. Configure an Ethernet connection with ethtool features
  hosts: node.example.com
  become: true
  tasks:
  - include_role:
      name: linux-system-roles.network

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

3. Run the playbook:

   - To connect as **root** user to the managed host, enter:

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

   - To connect as a user to the managed host, enter:

```
# ansible-playbook -u user_name --ask-become-pass ~/configure-ethernet-device-
with-ethtool-features.yml
```

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

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

Additional resources

- For a full list of **ethtool** features and details about the parameters used in **network_connections**, and for additional information about the **network** system role, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file.
- For details about the **ansible-playbook** command, see the **ansible-playbook(1)** man page.

CHAPTER 5. CONFIGURING SELINUX USING SYSTEM ROLES

5.1. INTRODUCTION TO THE SELINUX SYSTEM ROLE

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

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

The following table provides an overview of input variables available in the SELinux system role.

Table 5.1. SELinux system role variables

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

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

Additional resources

- For a detailed reference on SELinux role variables, install the **rhel-system-roles** package, and see the **README.md** or **README.html** files in the **/usr/share/doc/rhel-system-roles/selinux/** directory.
- For more information on RHEL System Roles, see Introduction to RHEL System Roles

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

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

Prerequisites

- Your Red Hat Ansible Engine subscription is attached to the system. See the How do I download and install Red Hat Ansible Engine article for more information.

Procedure

1. Enable the RHEL Ansible repository, for example:

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

2. Install Ansible Engine:

```
# yum install ansible
```

3. Install RHEL system roles:

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

4. Apply your playbook with an SELinux system role.
The following command applies an example playbook, which is a part of the **rhel-system-roles** package. You can use this playbook as a template:

```
# ansible-playbook -i host1,host2,host3 /usr/share/doc/rhel-system-roles/selinux/example-selinux-playbook.yml
```

Additional resources

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

CHAPTER 6. USING THE LOGGING SYSTEM ROLE

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

6.1. THE LOGGING SYSTEM ROLE

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

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

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

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

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

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

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

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

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

6.2. LOGGING SYSTEM ROLE PARAMETERS

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

NOTE

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

- **logging_inputs** - List of inputs for the logging solution.
  - **name** - Unique name of the input. Used in the **logging_flows** inputs list and a part of the generated **config** file name.

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

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

Additional resources

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

6.3. APPLYING A LOCAL LOGGING SYSTEM ROLE

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

Prerequisites

- You have Red Hat Ansible Engine installed on the system from which you want to run the playbook.

NOTE

You do not have to have Red Hat Ansible Engine installed on the systems on which you want to deploy the logging solution.

- You have the **rhel-system-roles** package on the system from which you want to run the playbook.

NOTE

You do not have to have **rsyslog** installed, because the system role installs **rsyslog** when deployed.

- You have an inventory file listing the systems on which you want to configure the logging solution.

Procedure

1. Create a playbook that defines the required role:

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

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

   b. Insert the following content:

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

2. Execute the playbook on a specific inventory:

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

   Where:

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

Verification

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

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

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

   a. Send a test message:

```
# logger test
```

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

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

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

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

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

Prerequisites

- You have Red Hat Ansible Engine installed on the system from which you want to run the playbook.

NOTE

You do not have to have Red Hat Ansible Engine installed on the systems on which you want to deploy the logging solution.

- You have the **rhel-system-roles** package on the system from which you want to run the playbook.

NOTE

You do not have to have **rsyslog** installed, because the system role installs **rsyslog** when deployed.

- You have at least two systems:
  - At least one will be the logging server.
  - At least one will be the logging client.

Procedure

1. **Create a playbook that defines the required role:**

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

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

   b. **Insert the following content into the file:**

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

   - name: Deploying basics input and forwards output
     hosts: clients
     roles:
       - linux-system-roles.logging
     vars:
       logging_inputs:
         - name: basic_input
           type: basics
       logging_outputs:
         - name: forward_output0
           type: forwards
           severity: info
           target: host1.example.com
           udp_port: 601
         - name: forward_output1
           type: forwards
           facility: mail
           target: host1.example.com
           tcp_port: 601
       logging_flows:
         - name: flows0
           inputs: [basic_input]
           outputs: [forward_output0, forward_output1]
   ```

```
[basic_input]
[forward_output0, forward_output1]
```

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

NOTE

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

WARNING

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

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

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

   ```
   # vi inventory.ini
   ```

   b. Insert the following content into the inventory file:

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

   Where: * ***host1.example.com*** is the logging server. ***host2.example.com*** is the logging client.

3. Execute the playbook on your inventory.

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

   Where:

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

Verification steps

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

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

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

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

   ```
   # logger test
   ```

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

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

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

Additional resources

- Getting started with RHEL System Roles
- Documentation installed with the **rhel-system-roles** package in **/usr/share/ansible/roles/rhel-system-roles.logging/README.html**
- RHEL System Roles KB article

6.5. ADDITIONAL RESOURCES

- Getting started with RHEL System Roles
- Documentation installed with the **rhel-system-roles** package in **/usr/share/ansible/roles/rhel-system-roles.logging/README.html**
- RHEL System Roles KB article

CHAPTER 7. USING THE CLEVIS AND TANG SYSTEM ROLES

7.1. INTRODUCTION TO THE CLEVIS AND TANG SYSTEM ROLES

RHEL System Roles is a collection of Ansible roles and modules that provide a consistent configuration interface to remotely manage multiple RHEL systems.

RHEL 8.3 introduced Ansible roles for automated deployments of Policy-Based Decryption (PBD) solutions using Clevis and Tang. The **rhel-system-roles** package contains these system roles, the related examples, and also the reference documentation.

The **nbde_client** system role enables you to deploy multiple Clevis clients in an automated way. Note that the **nbde_client** role supports only Tang bindings, and you cannot use it for TPM2 bindings at the moment.

Using the **nbde_server** role, you can deploy and manage a Tang server as part of an automated disk encryption solution. This role supports the following features:

- Rotating Tang keys
- Deploying and backing up Tang keys

Additional resources

- For a detailed reference on Network-Bound Disk Encryption (NBDE) role variables, install the **rhel-system-roles** package, and see the **README.md** and **README.html** files in the **/usr/share/doc/rhel-system-roles/nbde_client/** and **/usr/share/doc/rhel-system-roles/nbde_server/** directories.
- For example system-roles playbooks, install the **rhel-system-roles** package, and see the **/usr/share/ansible/roles/rhel-system-roles.nbde_server/examples/** directories.
- For more information on RHEL System Roles, see Introduction to RHEL System Roles

7.2. USING THE NBDE_SERVER SYSTEM ROLE FOR SETTING UP MULTIPLE TANG SERVERS

Follow the steps to prepare and apply an Ansible playbook containing your Tang-server settings.

Prerequisites

- Your Red Hat Ansible Engine subscription is attached to the system. See the How do I download and install Red Hat Ansible Engine article for more information.

Procedure

1. Enable the RHEL Ansible repository, for example:

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

2. Install Ansible Engine:

```
# yum install ansible
```

3. Install RHEL system roles:

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

4. Prepare your playbook containing settings for Tang servers. You can either start from the scratch, or use one of the example playbooks from the **/usr/share/ansible/roles/rhel-system-roles.nbde_server/examples/** directory.

```
# cp /usr/share/ansible/roles/rhel-system-roles.nbde_server/examples/simple_deploy.yml
./my-tang-playbook.yml
```

5. Edit the playbook in a text editor of your choice, for example:

```
# vi my-tang-playbook.yml
```

6. Add the required parameters. The following example playbook ensures deploying of your Tang server and a key rotation:

```
---
- hosts: all

  vars:
    nbde_server_rotate_keys: yes

  roles:
    - linux-system-roles.nbde_server
```

7. Apply the finished playbook:

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

Additional resources

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

7.3. USING THE NBDE_CLIENT SYSTEM ROLE FOR SETTING UP MULTIPLE CLEVIS CLIENTS

Follow the steps to prepare and apply an Ansible playbook containing your Clevis-client settings.

NOTE

The **nbde_client** system role supports only Tang bindings. This means that you cannot use it for TPM2 bindings at the moment.

Prerequisites

- Your Red Hat Ansible Engine subscription is attached to the system. See the How do I download and install Red Hat Ansible Engine article for more information.

Procedure

1. **Enable the RHEL Ansible repository, for example:**

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

2. **Install Ansible Engine:**

```
# yum install ansible
```

3. **Install RHEL system roles:**

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

4. **Prepare your playbook containing settings for Clevis clients. You can either start from the scratch, or use one of the example playbooks from the /usr/share/ansible/roles/rhel-system-roles.nbde_client/examples/ directory.**

```
# cp /usr/share/ansible/roles/rhel-system-roles.nbde_client/examples/high_availability.yml
./my-clevis-playbook.yml
```

5. **Edit the playbook in a text editor of your choice, for example:**

```
# vi my-clevis-playbook.yml
```

6. **Add the required parameters. The following example playbook configures Clevis clients for automated unlocking of two LUKS-encrypted volumes by when at least one of two Tang servers is available:**

```
---
- hosts: all

  vars:
    nbde_client_bindings:
      - device: /dev/rhel/root
        encryption_key_src: /etc/luks/keyfile
        servers:
          - http://server1.example.com
          - http://server2.example.com
      - device: /dev/rhel/swap
        encryption_key_src: /etc/luks/keyfile
        servers:
          - http://server1.example.com
          - http://server2.example.com

  roles:
    - linux-system-roles.nbde_client
```

7. **Apply the finished playbook:**

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

Additional resources

- For details about the parameters and additional information about the **nbde_client** role, install the **rhel-system-roles** package, and see the **/usr/share/doc/rhel-system-roles/nbde_client/** and **/usr/share/ansible/roles/rhel-system-roles.nbde_client/** directories.

CHAPTER 8. REQUESTING CERTIFICATES USING RHEL SYSTEM ROLES

With the Certificate System Role, you can use Red Hat Ansible Engine to issue and manage certificates.

This chapter covers the following topics:

- The Certificate System Role
- Requesting a new self-signed certificate using the Certificate System Role
- Requesting a new certificate from IdM CA using the Certificate System Role

8.1. THE CERTIFICATE SYSTEM ROLE

Using the Certificate System Role, you can manage issuing and renewing TLS and SSL certificates using Red Hat Ansible Engine.

The role uses **certmonger** as the certificate provider, and currently supports issuing and renewing self-signed certificates and using the IdM integrated certificate authority (CA).

You can use the following variables in your Ansible playbook with the Certificate System Role:

- certificate_wait to specify if the task should wait for the certificate to be issued.
- certificate_requests to represent each certificate to be issued and its parameters.

Additional resources

- For details about the parameters used in the **certificate_requests** variable and additional information about the **certificate** System Role, see the **/usr/share/ansible/roles/rhel-system-roles.certificate/README.md** file.
- For details about RHEL System Roles and how to apply them, see Getting started with RHEL System Roles.

8.2. REQUESTING A NEW SELF-SIGNED CERTIFICATE USING THE CERTIFICATE SYSTEM ROLE

With the Certificate System Role, you can use Red Hat Ansible Engine to issue self-signed certificates.

This process uses the **certmonger** provider and requests the certificate through the **getcert** command.

NOTE

By default, **certmonger** automatically tries to renew the certificate before it expires. You can disable this by setting the **auto_renew** parameter in the Ansible playbook to **no**.

Prerequisites

- You have Red Hat Ansible Engine installed on the system from which you want to run the playbook.

NOTE

You do not have to have Ansible installed on the systems on which you want to deploy the **certificate** solution.

- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.
  For details about RHEL System Roles and how to apply them, see Getting started with RHEL System Roles.

Procedure

1. *Optional:* Create an inventory file, for example **inventory.file**:

```
$ touch inventory.file
```

2. Open your inventory file and define the hosts on which you want to request the certificate, for example:

```
[webserver]
server.idm.example.com
```

3. Create a playbook file, for example **request-certificate.yml**:
   - Set **hosts** to include the hosts on which you want to request the certificate, such as **webserver**.
   - Set the **certificate_requests** variable to include the following:
     - Set the **name** parameter to the desired name of the certificate, such as **mycert**.
     - Set the **dns** parameter to the domain to be included in the certificate, such as ***.example.com**.
     - Set the **ca** parameter to **self-sign**.
   - Set the **rhel-system-roles.certificate** role under **roles**.
     This is the playbook file for this example:

```
---
- hosts: webserver

  vars:
    certificate_requests:
      - name: mycert
        dns: *.example.com
        ca: self-sign

  roles:
    - rhel-system-roles.certificate
```

4. Save the file.

5. Run the playbook:

```
$ ansible-playbook -i inventory.file request-certificate.yml
```

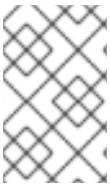
Additional resources

- For details about the parameters used in the **certificate_requests** variable and additional information about the **certificate** System Role, see the/**usr**/**share**/**ansible**/**roles**/**rhel-system-roles.certificate**/**README.md** file.
- For details about the **ansible-playbook** command, see the**ansible-playbook**(**1**) man page.

8.3. REQUESTING A NEW CERTIFICATE FROM IDM CA USING THE CERTIFICATE SYSTEM ROLE

With the Certificate System Role, you can use Red Hat Ansible Engine to issue certificates while using an IdM server with an integrated certificate authority (CA). Therefore, you can efficiently and consistently manage the certificate trust chain for multiple systems when using IdM as the CA.

This process uses the **certmonger** provider and requests the certificate through the**getcert** command.



NOTE

By default, **certmonger** automatically tries to renew the certificate before it expires. You can disable this by setting the **auto_renew** parameter in the Ansible playbook to **no**.

Prerequisites

- You have Red Hat Ansible Engine installed on the system from which you want to run the playbook.

NOTE

You do not have to have Ansible installed on the systems on which you want to deploy the **certificate** solution.

- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.
  For details about RHEL System Roles and how to apply them, see Getting started with RHEL System Roles.

Procedure

1. *Optional:* Create an inventory file, for example**inventory.file**:

```
$ touch inventory.file
```

2. Open your inventory file and define the hosts on which you want to request the certificate, for example:

```
[webserver]
server.idm.example.com
```

3. Create a playbook file, for example **request-certificate.yml**:
   - Set **hosts** to include the hosts on which you want to request the certificate, such as **webserver**.
   - Set the **certificate_requests** variable to include the following:
     - Set the **name** parameter to the desired name of the certificate, such as **mycert**.
     - Set the **dns** parameter to the domain to be included in the certificate, such as **www.example.com**.
     - Set the **principal** parameter to specify the Kerberos principal, such as **HTTP/www.example.com@EXAMPLE.COM**.
     - Set the **ca** parameter to **ipa**.
   - Set the **rhel-system-roles.certificate** role under **roles**.
     This is the playbook file for this example:

```
---
- hosts: webserver
  vars:
    certificate_requests:
      - name: mycert
        dns: www.example.com
        principal: HTTP/www.example.com@EXAMPLE.COM
        ca: ipa

  roles:
    - rhel-system-roles.certificate
```

4. Save the file.
5. Run the playbook:

```
$ ansible-playbook -i inventory.file request-certificate.yml
```

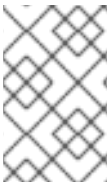
Additional resources

- For details about the parameters used in the **certificate_requests** variable and additional information about the **certificate** System Role, see the **/usr/share/ansible/roles/rhel-system-roles.certificate/README.md** file.
- For details about the **ansible-playbook** command, see the **ansible-playbook(1)** man page.

8.4. SPECIFYING COMMANDS TO RUN BEFORE OR AFTER CERTIFICATE ISSUANCE USING THE CERTIFICATE SYSTEM ROLE

With the Certificate System Role, you can use Red Hat Ansible Engine to execute a command before and after a certificate is issued or renewed.

In the following example, the administrator ensures stopping the **httpd** service before a self-signed certificate for **www.example.com** is issued or renewed, and restarting it afterwards.

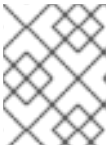


NOTE

By default, **certmonger** automatically tries to renew the certificate before it expires. You can disable this by setting the **auto_renew** parameter in the Ansible playbook to **no**.

Prerequisites

- You have Red Hat Ansible Engine installed on the system from which you want to run the playbook.



NOTE

You do not have to have Ansible installed on the systems on which you want to deploy the **certificate** solution.

- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.
  For details about RHEL System Roles and how to apply them, see Getting started with RHEL System Roles.

Procedure

1. *Optional:* Create an inventory file, for example **inventory.file**:

```
$ touch inventory.file
```

2. Open your inventory file and define the hosts on which you want to request the certificate, for example:

```
[webserver]
server.idm.example.com
```

3. Create a playbook file, for example **request-certificate.yml**:
   - Set **hosts** to include the hosts on which you want to request the certificate, such as **webserver**.
   - Set the **certificate_requests** variable to include the following:
     - Set the **name** parameter to the desired name of the certificate, such as **mycert**.
     - Set the **dns** parameter to the domain to be included in the certificate, such as **www.example.com**.
     - Set the **ca** parameter to the CA you want to use to issue the certificate, such as **self-sign**.
     - Set the **run_before** parameter to the command you want to execute before this certificate is issued or renewed, such as **systemctl stop httpd.service**.

- Set the **run_after** parameter to the command you want to execute after this certificate is issued or renewed, such as **systemctl start httpd.service**.

- Set the **rhel-system-roles.certificate** role under **roles**.
  This is the playbook file for this example:

```
---
- hosts: webserver
  vars:
    certificate_requests:
      - name: mycert
        dns: www.example.com
        ca: self-sign
        run_before: systemctl stop httpd.service
        run_after: systemctl start httpd.service

  roles:
    - linux-system-roles.certificate
```

4. Save the file.

5. Run the playbook:

```
$ ansible-playbook -i inventory.file request-certificate.yml
```

Additional resources

- For details about the parameters used in the **certificate_requests** variable and additional information about the **certificate** System Role, see the **/usr/share/ansible/roles/rhel-system-roles.certificate/README.md** file.

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

CHAPTER 9. CONFIGURING KDUMP USING RHEL SYSTEM ROLES

To manage kdump using Ansible, you can use the **kdump** role, which is one of the RHEL System Roles available in RHEL 8.

Using the **kdump** enables you to specify where to save the contents of the system's memory for later analysis.

For more information on RHEL System Roles and how to apply them, see Introduction to RHEL System Roles.

9.1. THE KDUMP RHEL SYSTEM ROLE

The **kdump** System Role enables you to set basic kernel dump parameters on multiple systems.

9.2. KDUMP ROLE PARAMETERS

The parameters used for the **kdump** RHEL System Roles are:

Role Variable	Description
kdump_path	The path to which **vmcore** is written. If **kdump_target** is not null, path is relative to that dump target. Otherwise, it must be an absolute path in the root file system.

Additional resources

- See the makedumpfile(8) man page.
- For details about the parameters used in **kdump** and additional information about the **kdump** System Role, see the **/usr/share/ansible/roles/rhel-system-roles.tlog/README.md** file.

9.3. CONFIGURING KDUMP USING RHEL SYSTEM ROLES

You can set basic kernel dump parameters on multiple systems using the **kdump** System Role by running an Ansible playbook.

WARNING

The **kdump** role replaces the kdump configuration of the managed hosts entirely by replacing the **/etc/kdump.conf** file. Additionally, if the kdump role is applied, all previous kdump settings are also replaced, even if they are not specified by the role variables, by replacing the **/etc/sysconfig/kdump** file.

Prerequisites

- You have Red Hat Ansible Engine installed on the system from which you want to run the playbook.

NOTE

You do not have to have Red Hat Ansible Automation Platform installed on the systems on which you want to deploy the **kdump** solution.

- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.
- You have an inventory file which lists the systems on which you want to deploy **kdump**.

Procedure

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

```
---
- hosts: kdump-test
  vars:
    kdump_path: /var/crash
  roles:
    - rhel-system-roles.kdump
```

2. Optional: Verify playbook syntax.

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

3. Run the playbook on your inventory file:

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

Additional resources

- For a detailed reference on kdump role variables, see the README.md or README.html files in the /usr/share/doc/rhel-system-roles/kdump directory.
- See Section 1.3, “Applying a role”.
- Documentation installed with the **rhel-system-roles** package/**usr/share/ansible/roles/rhel-system-roles.kdump/README.html**

CHAPTER 10. CONFIGURING STORAGE USING RHEL SYSTEM ROLES

To manage LVM and local file systems (FS) using Ansible, you can use the **storage** role, which is one of the RHEL System Roles available in RHEL 8.

Using the **storage role** enables you to automate administration of file systems on disks and logical volumes on multiple machines and across all versions of RHEL starting with RHEL 7.7.

For more information on RHEL System Roles and how to apply them, see Introduction to RHEL System Roles.

10.1. INTRODUCTION TO THE STORAGE ROLE

The **storage** role can manage:

- File systems on disks which have not been partitioned
- Complete LVM volume groups including their logical volumes and file systems

With the **storage** role you can perform the following tasks:

- Create a file system
- Remove a file system
- Mount a file system
- Unmount a file system
- Create LVM volume groups
- Remove LVM volume groups
- Create logical volumes
- Remove logical volumes
- Create RAID volumes
- Remove RAID volumes
- Create LVM pools with RAID
- Remove LVM pools with RAID

10.2. PARAMETERS THAT IDENTIFY A STORAGE DEVICE IN THE STORAGE SYSTEM ROLE

Your **storage** role configuration affects only the file systems, volumes, and pools that you list in the following variables.

storage_volumes

List of file systems on all unpartitioned disks to be managed.
Partitions are currently unsupported.

storage_pools

List of pools to be managed.
Currently the only supported pool type is LVM. With LVM, pools represent volume groups (VGs). Under each pool there is a list of volumes to be managed by the role. With LVM, each volume corresponds to a logical volume (LV) with a file system.

10.3. CREATING AN XFS FILE SYSTEM ON A BLOCK DEVICE USING RHEL SYSTEM ROLES

This section describes how to create an XFS file system on a block device on multiple target machines using the **storage** role.

Prerequisites

- An Ansible playbook that uses the **storage** role exists.
  For information on how to apply such a playbook, see Applying a role.

10.3.1. Example Ansible playbook to create an XFS file system on a block device

This section provides an example Ansible playbook. This playbook applies the **storage** role to create an XFS file system on a block device using the default parameters.

WARNING

The **storage** role can create a file system only on an unpartitioned, whole disk or a logical volume (LV). It cannot create the file system on a partition.

Example 10.1. A playbook that creates XFS on /dev/sdb

```
---
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: xfs
  roles:
    - rhel-system-roles.storage
```

- The volume name (***barefs*** in the example) is currently arbitrary. The**storage** role identifies the volume by the disk device listed under the **disks:** attribute.
- You can omit the **fs_type: xfs** line because XFS is the default file system in RHEL 8.

- To create the file system on an LV, provide the LVM setup under the **disks:** attribute, including the enclosing volume group. For details, see Example Ansible playbook to manage logical volumes.
  Do not provide the path to the LV device.

Additional resources

- For details about the parameters used in the **storage** system role, see the **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file.

10.4. PERSISTENTLY MOUNTING A FILE SYSTEM USING RHEL SYSTEM ROLES

This section describes how to persistently mount a file system using the **storage** role.

Prerequisites

- An Ansible playbook that uses the **storage** role exists.
  For information on how to apply such a playbook, see Applying a role.

10.4.1. Example Ansible playbook to persistently mount a file system

This section provides an example Ansible playbook. This playbook applies the **storage** role to immediately and persistently mount an XFS file system.

Example 10.2. A playbook that mounts a file system on /dev/sdb to /mnt/data

```
---
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: xfs
        mount_point: /mnt/data
  roles:
    - rhel-system-roles.storage
```

- This playbook adds the file system to the **/etc/fstab** file, and mounts the file system immediately.
- If the file system on the **/dev/sdb** device or the mount point directory do not exist, the playbook creates them.

Additional resources

- For details about the parameters used in the **storage** system role, see the **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file.

10.5. ENABLING ONLINE BLOCK DISCARD USING RHEL SYSTEM ROLES

This section describes how to enable online block discard using the **storage** role.

Prerequisites

- An Ansible playbook including the **storage** role exists.

For information on how to apply such a playbook, see Applying a role.

10.5.1. Example Ansible playbook to enable online block discard

This section provides an example Ansible playbook. This playbook applies the **storage** role to mount an XFS file system with online block discard enabled.

Example 10.3. A playbook that enables online block discard on /mnt/data/

```
---
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: xfs
        mount_point: /mnt/data
        mount_options: discard
  roles:
    - rhel-system-roles.storage
```

Additional resources

- This playbook also performs all the operations of the persistent mount example described in Example Ansible playbook to persistently mount a file system.
- For details about the parameters used in the **storage** system role, see the **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file.

10.6. CREATING AND MOUNTING EXT3 FILE SYSTEMS USING RHEL SYSTEM ROLES

This section describes how to create an ext3 file system with a given label on a disk, and persistently mount the file system using the **storage** role.

Prerequisites

- An Ansible playbook including the **storage** role exists.

For information on how to apply such a playbook, see Applying a role.

10.6.1. Example Ansible playbook to create and mount an ext3 file system

This section provides an example Ansible playbook. This playbook applies the **storage** role to create and mount an Ext3 file system.

Example 10.4. A playbook that creates Ext3 on /dev/sdb and mounts it at /mnt/data

```
---
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: ext3
        fs_label: label-name
        mount_point: /mnt/data
  roles:
    - rhel-system-roles.storage
```

- The playbook creates the file system on the /**dev**/**sdb** disk.
- The playbook persistently mounts the file system at the */**mnt**/**data*** directory.
- The label of the file system is ***label-name***.

Additional resources

- For details about the parameters used in the **storage** system role, see the /**usr**/**share**/**ansible**/**roles**/**rhel-system-roles.storage**/**README.md** file.

10.7. CREATING AND MOUNTING EXT4 FILE SYSTEMS USING RHEL SYSTEM ROLES

This section describes how to create an ext4 file system with a given label on a disk, and persistently mount the file system using the **storage** role.

Prerequisites

- An Ansible playbook including the **storage** role exists.

For information on how to apply such a playbook, see Applying a role.

10.7.1. Example Ansible playbook to create and mount an Ext4 file system

This section provides an example Ansible playbook. This playbook applies the **storage** role to create and mount an Ext4 file system.

Example 10.5. A playbook that creates Ext4 on /dev/sdb and mounts it at /mnt/data

```
---
```

```
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: ext4
        fs_label: label-name
        mount_point: /mnt/data
  roles:
    - rhel-system-roles.storage
```

- The playbook creates the file system on the **/dev/sdb** disk.
- The playbook persistently mounts the file system at the ***/mnt/data*** directory.
- The label of the file system is ***label-name***.

Additional resources

- For details about the parameters used in the **storage** system role, see the **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file.

10.8. MANAGING LVM LOGICAL VOLUMES USING RHEL SYSTEM ROLES

This section describes how to apply the **storage** role to perform the following tasks:

- Create an LVM logical volume in a volume group consisting of multiple disks.
- Create an ext4 file system with a given label on the logical volume.
- Persistently mount the ext4 file system.

Prerequisites

- An Ansible playbook including the **storage** role

For information on how to apply an Ansible playbook, see Applying a role.

10.8.1. Example Ansible playbook to manage logical volumes

This section provides an example Ansible playbook. This playbook applies the **storage** role to create an LVM logical volume in a volume group.

Example 10.6. A playbook that creates a mylv logical volume in the myvg volume group

```
- hosts: all
  vars:
    storage_pools:
      - name: myvg
        disks:
```

```
      - sda
      - sdb
      - sdc
    volumes:
      - name: mylv
        size: 2G
        fs_type: ext4
        mount_point: /mnt
roles:
  - rhel-system-roles.storage
```

- The **myvg** volume group consists of the following disks:
  - **/dev/sda**
  - **/dev/sdb**
  - **/dev/sdc**
- If the **myvg** volume group already exists, the playbook adds the logical volume to the volume group.
- If the **myvg** volume group does not exist, the playbook creates it.
- The playbook creates an Ext4 file system on the **mylv** logical volume, and persistently mounts the file system at **/mnt**.

Additional resources

- For details about the parameters used in the **storage** system role, see the **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file.

10.8.2. Additional resources

- For more information about the **storage** role, see Managing local storage using RHEL System Roles.

10.9. CONFIGURING A RAID VOLUME USING THE STORAGE SYSTEM ROLE

With the **storage** System Role, you can configure a RAID volume on RHEL using Red Hat Ansible Automation Platform. In this section you will learn how to set up an Ansible playbook with the available parameters to configure a RAID volume to suit your requirements.

Prerequisites

- You have Red Hat Ansible Engine installed on the system from which you want to run the playbook.

NOTE

You do not have to have Red Hat Ansible Automation Platform installed on the systems on which you want to deploy the **storage** solution.

- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.
- You have an inventory file detailing the systems on which you want to deploy a RAID volume using the **storage** System Role.

Procedure

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

```
- hosts: all
  vars:
    storage_safe_mode: false
    storage_volumes:
      - name: data
        type: raid
        disks: [sdd, sde, sdf, sdg]
        raid_level: raid0
        raid_chunk_size: 32 KiB
        mount_point: /mnt/data
        state: present
  roles:
    - name: rhel-system-roles.storage
```

WARNING

Device names can change in certain circumstances; for example, when you add a new disk to a system. Therefore, to prevent data loss, we do not recommend using specific disk names in the playbook.

2. Optional. Verify playbook syntax.

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

3. Run the playbook on your inventory file:

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

Additional resources

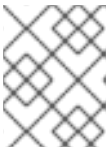
- For more information about RAID, see Managing RAID.
- For details about the parameters used in the storage system role, see the **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file.

10.10. CONFIGURING AN LVM POOL WITH RAID USING THE STORAGE SYSTEM ROLE

With the **storage** System Role, you can configure an LVM pool with RAID on RHEL using Red Hat Ansible Automation Platform. In this section you will learn how to set up an Ansible playbook with the available parameters to configure an LVM pool with RAID.

Prerequisites

- You have Red Hat Ansible Engine installed on the system from which you want to run the playbook.



NOTE

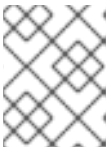
You do not have to have Red Hat Ansible Automation Platform installed on the systems on which you want to deploy the **storage** solution.

- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.
- You have an inventory file detailing the systems on which you want to configure an LVM pool with RAID using the **storage** System Role.

Procedure

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

```
- hosts: all
  vars:
    storage_safe_mode: false
    storage_pools:
      - name: my_pool
        type: lvm
        disks: [sdh, sdi]
        raid_level: raid1
        volumes:
          - name: my_pool
            size: "1 GiB"
            mount_point: "/mnt/app/shared"
            fs_type: xfs
            state: present
  roles:
    - name: rhel-system-roles.storage
```



NOTE

To create an LVM pool with RAID, you must specify the RAID type using the **raid_level** parameter.

2. Optional. Verify playbook syntax.

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

3. Run the playbook on your inventory file:

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

Additional resources

- For more information about RAID, see Managing RAID.
- For details about the parameters used in the storage system role, see the **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file.

10.11. MANAGING VOLUMES ENCRYPTED WITH LUKS USING RHEL SYSTEM ROLES

With the **storage** System Role, you can manage volumes encrypted with Linux Unified Key Setup-on-disk-format (LUKS) on RHEL using Red Hat Ansible Automation Platform.

10.11.1. Creating a LUKS encrypted volume using the storage role

You can use the **storage** role to create and configure a volume encrypted with LUKS by running an Ansible playbook.

Prerequisites

- You have Red Hat Ansible Engine installed on the system from which you want to run the playbook.

NOTE

You do not have to have Red Hat Ansible Automation Platform installed on the systems on which you want to create the volume.

- You have the **rhel-system-roles** package installed on the Ansible controller.
- You have an inventory file detailing the systems on which you want to deploy a LUKS encrypted volume using the storage System Role.

Procedure

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

```
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: xfs
        fs_label: label-name
        mount_point: /mnt/data
        encryption: true
        encryption_password: your-password
  roles:
    - rhel-system-roles.storage
```

2. Optional. Verify playbook syntax:

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

3. **Run the playbook on your inventory file:**

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

Additional resources

- For more information about LUKS, see 17. Encrypting block devices using LUKS..
- For details about the parameters used in the **storage** system role, see the **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file.

Additional resources

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

CHAPTER 11. CONFIGURING TIME SYNCHRONIZATION USING RHEL SYSTEM ROLES

With the **timesync** RHEL System Role, you can manage time synchronization on multiple target machines on RHEL using Red Hat Ansible Automation Platform.

11.1. THE TIMESYNC SYSTEM ROLE

You can manage time synchronization on multiple target machines using the **timesync** RHEL System Role.

The **timesync** role installs and configures an NTP or PTP implementation to operate as an NTP client or PTP replica in order to synchronize the system clock with NTP servers or grandmasters in PTP domains.

Note that using the **timesync** role also facilitates themigration to chrony, because you can use the same playbook on all versions of Red Hat Enterprise Linux starting with RHEL 6 regardless of whether the system uses **ntp** or **chrony** to implement the NTP protocol.

11.2. APPLYING THE TIMESYNC SYSTEM ROLE FOR A SINGLE POOL OF SERVERS

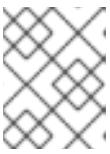
The following example shows how to apply the **timesync** role in a situation with just one pool of servers.

WARNING

The **timesync** role replaces the configuration of the given or detected provider service on the managed host. Previous settings are lost, even if they are not specified in the role variables. The only preserved setting is the choice of provider if the **timesync_ntp_provider** variable is not defined.

Prerequisites

- You have Red Hat Ansible Engine installed on the system from which you want to run the playbook.



NOTE

You do not have to have Red Hat Ansible Automation Platform installed on the systems on which you want to deploy the **timesync** solution.

- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.
- You have an inventory file which lists the systems on which you want to deploy **timesync** System Role.

Procedure

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

```
---
- hosts: timesync-test
  vars:
    timesync_ntp_servers:
      - hostname: 2.rhel.pool.ntp.org
        pool: yes
        iburst: yes
  roles:
    - rhel-system-roles.timesync
```

2. **Optional: Verify playbook syntax.**

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

3. **Run the playbook on your inventory file:**

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

11.3. TIMESYNC SYSTEM ROLES VARIABLES

You can pass the following variable to the timesync role:

- **timesync_ntp_servers:**

Role variable settings	Description
hostname: host.example.com	Hostname or address of the server
minpoll: *number*	Minimum polling interval. Default: 6
maxpoll: *number*	Maximum polling interval. Default: 10
iburst: yes	Flag enabling fast initial synchronization. Default: no
pool: yes	Flag indicating that each resolved address of the hostname is a separate NTP server. Default: no

Additional resources

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

CHAPTER 12. MONITORING PERFORMANCE USING RHEL SYSTEM ROLES

12.1. INTRODUCTION TO THE METRICS SYSTEM ROLE

RHEL System Roles is a collection of Ansible roles and modules that provide a consistent configuration interface to remotely manage multiple RHEL systems. The metrics System Role configures performance analysis services for the local system and, optionally, includes a list of remote systems to be monitored by the local system. The metrics System Role enables you to use pcp to monitor your systems performance without having to configure pcp separately, as the set-up and deployment of pcp is handled by the playbook.

Table 12.1. Metrics system role variables

Role variable	Description	Example usage
metrics_monitored_hosts	List of remote hosts to be analyzed by the target host. These hosts will have metrics recorded on the target host, so ensure enough disk space exists below **/var/log** for each host.	**metrics_monitored_hosts: ["*webserver.example.com*", "*database.example.com*"]**
metrics_retention_days	Configures the number of days for performance data retention before deletion.	**metrics_retention_days: 14**
metrics_graph_service	A boolean flag that enables the host to be set up with services for performance data visualization via **pcp** and **grafana**. Set to false by default.	**metrics_graph_service: false**
metrics_query_service	A boolean flag that enables the host to be set up with time series query services for querying recorded **pcp** metrics via **redis**. Set to false by default.	**metrics_query_service: false**
metrics_provider	Specifies which metrics collector to use to provide metrics. Currently, **pcp** is the only supported metrics provider.	**metrics_provider: "pcp"**

Additional resources

- for details about the parameters used in **metrics_connections** and additional information about the metrics System Role, see the **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** file.

12.2. USING THE METRICS SYSTEM ROLE TO MONITOR YOUR LOCAL SYSTEM WITH VISUALIZATION

This procedure describes how to use the metrics RHEL System Role to monitor your local system while simultaneously provisioning data visualization via **grafana**.

Prerequisites

- You have Red Hat Ansible Engine installed on the machine you want to monitor.
- You have the **rhel-system-roles** package installed on the machine you want to monitor.

Procedure

1. Configure **localhost** in the the/**etc**/**ansible**/**hosts** Ansible inventory by adding the following content to the inventory:

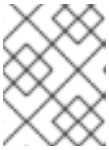
```
localhost ansible_connection=local
```

2. Create an Ansible playbook with the following content:

```
---
- hosts: localhost
  vars:
    metrics_graph_service: yes
  roles:
    - rhel-system-roles.metrics
```

3. Run the Ansible playbook:

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



NOTE

Since the **metrics_graph_service** boolean is set to value="yes",**grafana** is automatically installed and provisioned with **pcp** added as a data source.

4. To view visualization of the metrics being collected on your machine, access the **grafana** web interface as described in Accessing the Grafana web UI.

12.3. USING THE METRICS SYSTEM ROLE TO SETUP A FLEET OF INDIVIDUAL SYSTEMS TO MONITOR THEMSELVES

This procedure describes how to use the metrics System Role to set up a fleet of machines to monitor themselves.

Prerequisites

- You have Red Hat Ansible Engine installed on the machine you want to use to run the playbook.

- You have the **rhel-system-roles** package installed on the machine you want to use to run the playbook.

Procedure

1. Add the name or IP of the machines you wish to monitor via the playbook to the **/etc/ansible/hosts** Ansible inventory file under an identifying group name enclosed in brackets:

```
[remotes]
webserver.example.com
database.example.com
```

2. Create an Ansible playbook with the following content:

```
---
- hosts: remotes
  vars:
    metrics_retention_days: 0
  roles:
    - rhel-system-roles.metrics
```

3. Run the Ansible playbook:

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

12.4. USING THE METRICS SYSTEM ROLE TO MONITOR A FLEET OF MACHINES CENTRALLY VIA YOUR LOCAL MACHINE

This procedure describes how to use the metrics System Role to set up your local machine to centrally monitor a fleet of machines while also provisioning visualization of the data via **grafana** and querying of the data via **redis**.

Prerequisites

- You have Red Hat Ansible Engine installed on the machine you want to use to run the playbook.
- You have the **rhel-system-roles** package installed on the machine you want to use to run the playbook.

Procedure

1. Create an Ansible playbook with the following content:

```
---
- hosts: localhost
  vars:
    metrics_graph_service: yes
    metrics_query_service: yes
    metrics_retention_days: 10
```

```
    metrics_monitored_hosts: ["database.example.com", "webserver.example.com"]
  roles:
    - rhel-system-roles.metrics
```

2. **Run the Ansible playbook:**

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

NOTE

Since the **metrics_graph_service** and **metrics_query_service** booleans are set to value="yes", **grafana** is automatically installed and provisioned with **pcp** added as a data source with the **pcp** data recording indexed into **redis**, allowing the **pcp** querying language to be used for complex querying of the data.

3. To view graphical representation of the metrics being collected centrally by your machine and to query the data, access the **grafana** web interface as described in Accessing the Grafana web UI.

CHAPTER 13. CONFIGURING A SYSTEM FOR SESSION RECORDING USING THE TLOG RHEL SYSTEM ROLES

With the **tlog** RHEL System Role, you can configure a system for terminal session recording on RHEL using Red Hat Ansible Automation Platform.

13.1. THE TLOG SYSTEM ROLE

You can configure a RHEL system for terminal session recording on RHEL using the **tlog** RHEL System Role. The **tlog** package and its associated web console session player provide you with the ability to record and play back user terminal sessions.

You can configure the recording to take place per user or user group via the **SSSD** service. All terminal input and output is captured and stored in a text-based format in the system journal.

Additional resources

- For more details on session recording in RHEL, see Recording Sessions

13.2. COMPONENTS AND PARAMETERS OF THE TLOG SYSTEM ROLES

The Session Recording solution is composed of the following components:

- The tlog utility
- System Security Services Daemon (SSSD)
- Optional: The web console interface

The parameters used for the tlog RHEL System Roles are:

Role Variable	Description
tlog_use_sssd (default: yes)	Configure session recording with SSSD, the preferred way of managing recorded users or groups
tlog_scope_sssd (default: none)	Configure SSSD recording scope - all / some / none
tlog_users_sssd (default: [])	YAML list of users to be recorded
tlog_groups_sssd (default: [])	YAML list of groups to be recorded

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

13.3. DEPLOYING THE TLOG RHEL SYSTEM ROLE

Follow these steps to prepare and apply an Ansible playbook to configure a RHEL system to log recording data to the systemd journal.

Prerequisites

- You have set SSH keys for access from the control node to the target system where the **tlog** System Role will be configured.
- You have one control node, which is a system from which the Ansible Engine configures the other systems.
- You have Red Hat Ansible Engine installed on the control node, from which you want to run the playbook.
- You have the **rhel-system-roles** package installed on the control node from which you want to run the playbook.
- You have at least one system that you want to configure the **tlog** System Role. You do not have to have Red Hat Ansible Automation Platform installed on the systems on which you want to deploy the **tlog** solution.

Procedure

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

```
---
- name: Deploy session recording
  hosts: all
  vars:
    tlog_scope_sssd: some
    tlog_users_sssd:
      - recordeduser

  roles:
    - rhel-system-roles.tlog
```

Where,

- **tlog_scope_sssd**:
  - **some** specifies you want to record only certain users and groups, not **all** or **none**.
- **tlog_users_sssd**:
  - **recordeduser** specifies the user you want to record a session from. Note that this does not add the user for you. You must set the user by yourself.

2. Optionally, verify the playbook syntax.

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

3. Run the playbook on your inventory file:

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

As a result, the playbook installs the **tlog** role on the system you specified. It also creates an SSSD configuration drop file that can be used by the users and groups that you define. SSSD parses and reads these users and groups to overlay **tlog** session as the shell user. Additionally, if the **cockpit**

package is installed on the system, the playbook also installs the **cockpit-session-recording** package, which is a **Cockpit** module that allows you to view and play recordings in the web console interface.

Verification steps

To verify that the SSSD configuration drop file is created in the system, perform the following steps:

1. Navigate to the folder where the SSSD configuration drop file is created:

```
# cd /etc/sssd/conf.d
```

2. Check the file content:

```
# cat /etc/sssd/conf.d/sssd-session-recording.conf
```

You can see that the file contains the parameters you set in the playbook.

13.4. RECORDING A SESSION USING THE DEPLOYED TLOG SYSTEM ROLE IN THE CLI

Once you have deployed the **tlog** System Role in the system you have specified, you are able to record a user terminal session using the command-line interface (CLI).

Prerequisites

- You have deployed the **tlog** System Role in the target system.
- The SSSD configuration drop file was created in the **/etc/sssd/conf.d** file.

Procedure

1. Create a user and assign a password for this user:

```
# useradd recordeduser
# passwd recordeduser
```

2. Relog to the system as the user you just created:

```
# ssh recordeduser@localhost
```

3. Type "yes" when the system prompts you to type yes or no to authenticate.
4. Insert the *recordeduser's* password.
The system prompts a message to inform that your session is being recorded.

```
ATTENTION! Your session is being recorded!
```

5. Once you have finished recording the session, type:

```
# exit
```

The system logs out from the user and closes the connection with the localhost.

As a result, the user session is recorded, stored and you can play it using a journal.

Verification steps

To view your recorded session in the journal, do the following steps:

1. Run the command below:

```
# journalctl -o verbose -r
```

2. Search for the **MESSAGE** field of the**tlog-rec** recorded journal entry.

13.5. WATCHING A RECORDED SESSION USING THE CLI

You can play a user session recording from a journal using the command-line interface (CLI).

Prerequisites

- You have recorded a user session. See Section 13.4, "Recording a session using the deployed tlog system role in the CLI"

Procedure

1. On the CLI terminal, play the user session recording:

```
# journalctl -o verbose -r
```

2. Search for the **tlog** recording:

```
$ /tlog-rec
```

You can see details such as:

- The username for the user session recording
- The **out_txt** field, a raw output encode of the recorded session
- The identifier number TLOG_REC=*ID_number*

3. Copy the identifier number TLOG_REC=*ID_number*.
4. Playback the recording using the identifier number TLOG_REC=*ID_number*.

```
# tlog-play -r journal -M TLOG_REC=ID_number
```

As a result, you can see the user session recording terminal output being played back.