Red Hat Enterprise Linux 9.0 Beta

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: 2021-11-16

Red Hat Enterprise Linux 9.0 Beta 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 9. 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 RHEL 9.

Table of Contents

RHEL BETA RELEASE

Red Hat provides Red Hat Enterprise Linux Beta access to all subscribed Red Hat accounts. The purpose of Beta access is to:

- Provide an opportunity to customers to test major features and capabilities prior to the general availability release and provide feedback or report issues.
- Provide Beta product documentation as a preview. Beta product documentation is under development and is subject to substantial change.

Note that Red Hat does not support the usage of RHEL Beta releases in production use cases. For more information, see What does Beta mean in Red Hat Enterprise Linux and can I upgrade a RHEL Beta installation to a General Availability (GA) release?.

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 9, the interface currently consists of the following roles:

- network
- certificate
- postfix
- kernel_settings
- metrics
- nbde_client and nbde_server
- tlog
- ssh
- sshd
- crypto_policies

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

Additional resources

- Red Hat Enterprise Linux (RHEL) System Roles
- **/usr/share/doc/rhel-system-roles** documentation [1]
- Administration and configuration tasks using System Roles in RHEL

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

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

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

1. Install the Ansible Core package:

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

The Ansible Core package provides the **ansible-playbook** CLI, the Ansible Vault functionality, and the basic modules and filters required by RHEL Ansible content.

- Ensure that you are able to create an Ansible inventory.
Inventories represent the hosts, host groups, and some of the configuration parameters used by the Ansible playbooks.

  Playbooks are typically human-readable, and are defined in **ini**, **yaml**, **json**, and other file formats.

- Ensure that 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 are defined in the **yaml** format.

Procedure

1. Create the required Ansible inventory containing the hosts and groups that you want to manage. Here is an example using a file called **inventory.ini** of a group of hosts called **webservers**:

```
[webservers]
```

```
host1
host2
host3
```

2. Create an Ansible playbook including the required role. The following example shows how to use roles through the **roles:** option for a playbook:
   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.postfix
```

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 **postfix**, **metrics**, **network**, **tlog**, or **ssh**.

3. To execute the playbook on specific hosts, you must perform one of the following:

   - Edit the playbook to use **hosts: host1[,host2,...]**, or **hosts: all**, and execute the command:

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

   - Edit the inventory to ensure that the hosts you want to use are defined in a group, and execute the command:

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

   - Specify all hosts when executing the **ansible-playbook** command:

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

IMPORTANT

Be aware that the **-i** flag specifies the inventory of all hosts that are available. If you have multiple targeted hosts, but want to select a host against which you want to run the playbook, you can add a variable in the playbook to be able to select a host. For example:

```
Ansible Playbook | example-playbook.yml:


- hosts: "{{ target_host }}"
  roles:
     - rhel-system-roles.network
     - rhel-system-roles.postfix
```

Playbook execution command:

```
# ansible-playbook -i host1,..hostn -e target_host=host5 example-
playbook.yml
```

Additional resources

- Ansible playbooks
- Using roles in Ansible playbook
- Examples of Ansible playbooks
- How to create and work with inventory?
- ansible-playbook

1.4. ADDITIONAL RESOURCES

- Red Hat Enterprise Linux (RHEL) System Roles Red Hat Knowledgebase article

[1] This documentation is installed automatically with the **rhel-system-roles** package.

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

To use the RHEL System Roles, install the required packages in your system.

Prerequisites

- 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
```

2. Install the Ansible Core package:

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

The Ansible Core package provides the **ansible-playbook** CLI, the Ansible Vault functionality, and the basic modules and filters required by RHEL Ansible content.

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

Additional resources

- The Red Hat Enterprise Linux (RHEL) System Roles
- The **ansible-playbook** man page.

CHAPTER 3. INSTALLING AND USING COLLECTIONS

3.1. INTRODUCTION TO ANSIBLE COLLECTIONS

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

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

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

Additional resources

- You can find the Red Hat Certified Collections by accessing the Automation Hub.

3.2. COLLECTIONS STRUCTURE

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

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

3.3. INSTALLING COLLECTIONS BY USING THE CLI

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

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

Prerequisites

- The **python3-jmespath** package is installed.
- An inventory file that lists the managed nodes exists.

Procedure

- Install the collection via RPM package:

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

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

Verification steps

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

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

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

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

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

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

Additional resources

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

3.4. INSTALLING COLLECTIONS FROM AUTOMATION HUB

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

Prerequisites

- The **python3-jmespath** package is installed.

- An inventory file that lists the managed nodes exists.

Procedure

1. Install the **redhat.rhel_system_roles** collection from the Automation Hub:

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

2. Define Red Hat Automation Hub as the default source for content in the **ansible.cfg** configuration file. See Configuring Red Hat Automation Hub as the primary source for content . After the installation is finished, the roles are available as **redhat.rhel_system_roles.<role_name>**. Additionally, you can find the documentation for each role at **/usr/share/ansible/collections/ansible_collections/redhat/rhel_system_roles/roles/<role_name>/README.md**.

Verification steps

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

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

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

2. Edit the file, replacing "hosts: all" with "hosts: localhost" to make the playbook run only on the local system.

3. Run the ansible-playbook on the check mode. This does not change any settings on your system.

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

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

Additional resources

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

3.5. DEPLOYING THE TLOG RHEL SYSTEM ROLE USING COLLECTIONS

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

Prerequisites

- A Galaxy collection is installed.

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:
    - redhat.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

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:

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

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

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

2. Check the file content:

```
# cat sssd-session-recording.conf
```

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

CHAPTER 4. USING ANSIBLE ROLES TO PERMANENTLY CONFIGURE KERNEL PARAMETERS

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.

4.1. 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 Core to configure according to the playbook.

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

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

Additional resources

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

4.2. 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

- The Ansible Core package is installed on the control machine.
- The **rhel-system-roles** package is installed on the control machine.

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 Core more effectively against a specific collection of systems.

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

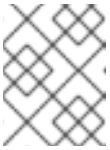
b. Insert the following content into the file:

```
---
- 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 Core 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

- Getting started with RHEL System Roles
- **README.html** and **README.md** files in the **/usr/share/doc/rhel-system-roles/kernel_settings/** directory
- Working with Inventory
- Configuring Ansible
- Working With Playbooks
- Using Variables
- Roles

CHAPTER 5. 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.

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

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

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

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

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

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

You can identify the device path with the following command:

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

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

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

Run this procedure on the Ansible control node.

Prerequisites

- The Ansible Core package 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: example
   match:
     path:
       - pci-0000:00:0[1-3].0
       - &!pci-0000:00:02.0
         type: ethernet
         autoconnect: yes
         ip:
           address:
             - 192.0.2.1/24
             - 2001:db8:1::1/64
```

```
      gateway4: 192.0.2.254
      gateway6: 2001:db8:1::fffe
      dns:
        - 192.0.2.200
        - 2001:db8:1::ffbb
      dns_search:
        - example.com
    state: up
```

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

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

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

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

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

Prerequisites

- A DHCP server is available in the network.
- The Ansible Core package 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
  interface_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

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

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

This procedure describes how to use RHEL System Roles to remotely add a dynamic Ethernet

connection for devices that match a specific device path by running an Ansible playbook. With dynamic IP settings, the network connection requests the IP settings for this connection from a DHCP server. Run this procedure on the Ansible control node.

You can identify the device path with the following command:

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

Prerequisites

- A DHCP server is available in the network.
- The Ansible Core package 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: example
       match:
         path:
           - pci-0000:00:0[1-3].0
           - &!pci-0000:00:02.0
             type: ethernet
             autoconnect: yes
             ip:
               dhcp4: yes
               auto6: yes
             state: up
   ```

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

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

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

5.5. 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 Core package 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

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

5.6. 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 Core package 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
  controller: bridge0
  port_type: bridge
  state: up

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

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.

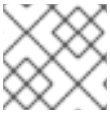
   If you do not specify the **-u *user_name*** option, **ansible-playbook** connects to the managed host as the user that is currently logged in to the control node.

Additional resources

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

5.7. CONFIGURING A NETWORK BOND USING RHEL SYSTEM ROLES

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



NOTE

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

Prerequisites

- The **ansible-core** package 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
    controller: bond0
    state: up

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

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

5.8. 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 Core package 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 is stored in the **/srv/data/client.key** file.
  - The client certificate is stored in the **/srv/data/client.crt** file.
  - The Certificate Authority (CA) certificate is stored in the **/srv/data/ca.crt** file.

Procedure

1. 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.

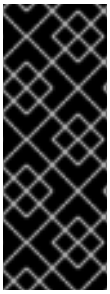
   If you do not specify the **-u *user_name*** option, **ansible-playbook** connects to the managed host as the user that is currently logged in to the control node.

Additional resources

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

5.9. 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 Core package 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

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

5.10. 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 Core package 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:

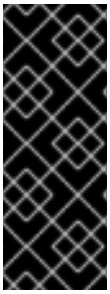
```
# ip -4 route
default via 198.51.100.254 dev enp7s0 proto static metric 100
192.0.2.0/24 via 198.51.100.1 dev enp7s0 proto static metric 100
203.0.113.0/24 via 198.51.100.2 dev enp7s0 proto static metric 100
...
```

Additional resources

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

5.11. 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-core** package 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

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

5.12. USING SYSTEM ROLES TO CONFIGURE ETHTOOL COALESCE SETTINGS

You can use the **networking** RHEL System Role to configure **ethtool** coalesce settings 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** coalesce settings:
  - RX frames: **128**
  - TX frames: **128**

Prerequisites

- The Ansible Core package 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-ethtoolcoalesce-settings.yml** playbook with the following content:

```
---
- name: Configure an Ethernet connection with ethtool coalesce settings
```

```
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:
          coalesce:
            rx_frames: 128
            tx_frames: 128
        state: up
```

3. Run the playbook:

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

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

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

     ```
     # ansible-playbook -u user_name --ask-become-pass ~/configure-ethernet-device-with-ethtoolcoalesce-settings.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

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

CHAPTER 6. CONFIGURING SECURE COMMUNICATION WITH THE SSH SYSTEM ROLES

As an administrator, you can use the SSHD System Role to configure SSH servers and the SSH System Role to configure SSH clients consistently on any number of RHEL systems at the same time using the Ansible Core package.

6.1. SSHD SYSTEM ROLE VARIABLES

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

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

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

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

renders as:

```
ListenAddress 0.0.0.0
ListenAddress ::
```

Variables for the SSHD System Role

sshd_enable

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

sshd_skip_defaults

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

sshd_manage_service

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

sshd_allow_reload

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

sshd_install_service

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

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

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

sshd_service

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

sshd

A dict that contains configuration. For example:

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

sshd_*OptionName*

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

```
sshd_Compression: no
```

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

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

Secondary variables for the SSHD System Role

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

sshd_packages

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

sshd_config_owner, **sshd_config_group**, and **sshd_config_mode**

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

sshd_config_file

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

sshd_binary

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

sshd_service

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

sshd_verify_hostkeys

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

sshd_hostkey_owner, sshd_hostkey_group, sshd_hostkey_mode

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

sshd_sysconfig

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

sshd_sysconfig_override_crypto_policy

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

sshd_sysconfig_use_strong_rng

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

6.2. CONFIGURING OPENSSH SERVERS USING THE SSHD SYSTEM ROLE

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

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the SSHD System Role.
- The **ansible-core** package is installed on the control machine.

Procedure

1. Copy the example playbook for the SSHD System Role:

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

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

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

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

```
    - Condition: "Address 192.0.2.0/24"
      PermitRootLogin: yes
      PasswordAuthentication: yes
```

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

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

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

3. **Optional: Verify playbook syntax.**

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

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

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

...

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

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

Verification

1. **Log in to the SSH server:**

```
$ ssh user1@10.1.1.1
```

Where:

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

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

```
$ vim /etc/ssh/sshd_config

# Ansible managed
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_ecdsa_key
HostKey /etc/ssh/ssh_host_ed25519_key
AcceptEnv LANG LC_CTYPE LC_NUMERIC LC_TIME LC_COLLATE LC_MONETARY
LC_MESSAGES
AcceptEnv LC_PAPER LC_NAME LC_ADDRESS LC_TELEPHONE LC_MEASUREMENT
AcceptEnv LC_IDENTIFICATION LC_ALL LANGUAGE
AcceptEnv XMODIFIERS
AuthorizedKeysFile .ssh/authorized_keys
```

```
ChallengeResponseAuthentication no
GSSAPIAuthentication yes
GSSAPICleanupCredentials no
PasswordAuthentication no
PermitRootLogin no
PrintMotd no
Subsystem sftp /usr/libexec/openssh/sftp-server
SyslogFacility AUTHPRIV
UsePAM yes
X11Forwarding yes
Match Address 192.0.2.0/24
  PasswordAuthentication yes
  PermitRootLogin yes
```

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

   a. **Determine your IP address:**

   ```
   $ hostname -I
   192.0.2.1
   ```

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

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

   ```
   $ ssh root@10.1.1.1
   ```

Additional resources

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

6.3. SSH SYSTEM ROLE VARIABLES

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

If you do not configure these variables, the system role produces a global ssh_config file that matches the RHEL defaults.

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

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

renders as:

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

NOTE

The configuration options are case sensitive.

Variables for the SSH System Role

ssh_user

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

ssh_skip_defaults

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

ssh_drop_in_name

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

WARNING

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

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

ssh

A dict that contains configuration options and their respective values.

ssh_*OptionName*

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

ssh_additional_packages

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

ssh_config_file

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

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

- if the **ssh_user** variable is defined, the default value is ~/**.ssh/config**.

ssh_config_owner, ssh_config_group, ssh_config_mode

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

6.4. CONFIGURING OPENSSH CLIENTS USING THE SSH SYSTEM ROLE

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

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the SSH System Role.
- The Ansible Core package is installed on the control machine.

Procedure

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

```
---
- hosts: all
  tasks:
  - name: "Configure ssh clients"
    include_role:
      name: rhel-system-roles.ssh
    vars:
      ssh_user: root
      ssh:
        Compression: true
        GSSAPIAuthentication: no
        ControlMaster: auto
        ControlPath: ~/.ssh/.cm%C
        Host:
          - Condition: example
            Hostname: example.com
            User: user1
      ssh_ForwardX11: no
```

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

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

- X11 forwarding is disabled.

Optionally, you can modify these variables according to your preferences. For more details, see SSH Client Role variables.

2. Optional: Verify playbook syntax.

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

3. Run the playbook on your inventory file:

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

Verification

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

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

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

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

CHAPTER 7. SETTING A CUSTOM CRYPTOGRAPHIC POLICY ACROSS SYSTEMS

As an administrator, you can use the Crypto Policies System Role on RHEL to quickly and consistently configure custom cryptographic policies across many different systems using the Ansible Core package.

7.1. CRYPTO POLICIES SYSTEM ROLE VARIABLES AND FACTS

In a Crypto Policies System Role playbook, you can define the parameters for the crypto policies configuration file according to your preferences and limitations.

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

Selected variables for the Crypto Policies System Role

crypto_policies_policy

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

crypto_policies_reload

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

crypto_policies_reboot_ok

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

Facts set by the Crypto Policies System Role

crypto_policies_active

Lists the currently selected policy.

crypto_policies_available_policies

Lists all available policies available on the system.

crypto_policies_available_subpolicies

Lists all available subpolicies available on the system.

Additional resources

- Creating and setting a custom system-wide cryptographic policy

7.2. SETTING A CUSTOM CRYPTOGRAPHIC POLICY USING THE CRYPTO POLICIES SYSTEM ROLE

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

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the Crypto Policies System Role.
- The Ansible Core package is installed on the control machine.

Procedure

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

```
---
- hosts: all
  tasks:
  - name: Configure crypto policies
    include_role:
      name: linux-system-roles.crypto_policies
    vars:
      - crypto_policies_policy: FUTURE
      - crypto_policies_reboot_ok: true
```

   You can replace the *FUTURE* value with your preferred crypto policy, for example: **DEFAULT**, **LEGACY**, and **FIPS:OSPP**.

   The **crypto_policies_reboot_ok: true** variable causes the system to reboot after the system role changes the crypto policy.

   For more details, see Crypto Policies System Role variables and facts.

2. Optional: Verify playbook syntax.

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

3. Run the playbook on your inventory file:

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

Verification

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

```
- hosts: all
  tasks:
  - name: Verify active crypto policy
    include_role:
      name: linux-system-roles.crypto_policies

  - debug:
      var: crypto_policies_active
```

   This playbook does not change any configurations on the system, only reports the active policy on the managed nodes.

2. Run the playbook on the same inventory file:

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

```
TASK [debug] *************************
ok: [host] => {
    "crypto_policies_active": "FUTURE"
}
```

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

7.3. ADDITIONAL RESOURCES

- **/usr/share/ansible/roles/rhel-system-roles.crypto_policies/README.md** file.
- **ansible-playbook(1)** man page.
- Installing RHEL System Roles. .
- Applying a system role .

CHAPTER 8. USING THE CLEVIS AND TANG SYSTEM ROLES

8.1. 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.

You can use 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.

The **nbde_client** role requires volumes that are already encrypted using LUKS. This role supports to bind a LUKS-encrypted volume to one or more Network-Bound (NBDE) servers - Tang servers. You can either preserve the existing volume encryption with a passphrase or remove it. After removing the passphrase, you can unlock the volume only using NBDE. This is useful when a volume is initially encrypted using a temporary key or password that you should remove after the system you provision the system.

If you provide both a passphrase and a key file, the role uses what you have provided first. If it does not find any of these valid, it attempts to retrieve a passphrase from an existing binding.

PBD defines a binding as a mapping of a device to a slot. This means that you can have multiple bindings for the same device. The default slot is slot 1.

The **nbde_client** role provides also the **state** variable. Use the **present** value for either creating a new binding or updating an existing one. Contrary to a **clevis luks bind** command, you can use **state: present** also for overwriting an existing binding in its device slot. The **absent** value removes a specified binding.

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

- Rotating Tang keys
- Deploying and backing up Tang keys

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

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

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

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **nbde_server** System Role.
- The **ansible-core** package is installed on the control machine.
- The **rhel-system-roles** package is installed on the system from which you want to run the playbook.

Procedure

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

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

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

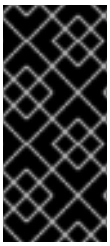
```
# vi my-tang-playbook.yml
```

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

```
---
- hosts: all

  vars:
    nbde_server_rotate_keys: yes

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

4. Apply the finished playbook:

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



IMPORTANT

To ensure that networking for a Tang pin is available during early boot by using the **grubby** tool on the systems where Clevis is installed:

```
# grubby --update-kernel=ALL --args="rd.neednet=1"
```

Additional resources

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

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

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

NOTE

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

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **nbde_client** System Role.
- The Ansible Core package is installed on the control machine.
- The **rhel-system-roles** package is installed on the system from which you want to run the playbook.

Procedure

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

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

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

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

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

```
---
- hosts: all

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

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

4. **Apply the finished playbook:**

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

IMPORTANT

To ensure that networking for a Tang pin is available during early boot by using the grubby tool on the system where Clevis is installed:

```
# grubby --update-kernel=ALL --args="rd.neednet=1"
```

Additional resources

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

CHAPTER 9. REQUESTING CERTIFICATES USING RHEL SYSTEM ROLES

You can use the Certificate System Role 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

9.1. THE CERTIFICATE SYSTEM ROLE

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

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

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

certificate_wait

to specify if the task should wait for the certificate to be issued.

certificate_requests

to represent each certificate to be issued and its parameters.

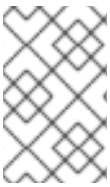
Additional resources

- 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.

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

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

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



NOTE

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

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.
  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
```

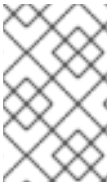
Additional resources

- 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.

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

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

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



NOTE

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

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.
  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
```

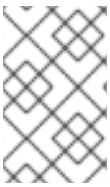
Additional resources

- 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.

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

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

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



NOTE

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

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.
  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 10. MONITORING PERFORMANCE USING RHEL SYSTEM ROLES

As a system administrator, you can use the metrics RHEL System Role to monitor the performance of a system.

10.1. 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 10.1. 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"**

NOTE

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

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

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

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the **rhel-system-roles** package installed on the machine you want to monitor.

Procedure

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

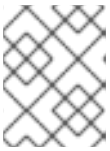
```
localhost ansible_connection=local
```

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.

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

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

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the **rhel-system-roles** package installed on the machine you want to use to run the playbook.

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
```

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

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

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the **rhel-system-roles** package installed on the machine you want to use to run the playbook.

Procedure

1. Create an Ansible playbook with the following content:

```
---
- hosts: localhost
  vars:
    metrics_graph_service: yes
    metrics_query_service: yes
    metrics_retention_days: 10
    metrics_monitored_hosts: ["database.example.com", "webserver.example.com"]
  roles:
    - rhel-system-roles.metrics
```

2. Run the Ansible playbook:

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

NOTE

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

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

10.5. SETTING UP AUTHENTICATION WHILE MONITORING A SYSTEM USING THE METRICS SYSTEM ROLE

PCP supports the **scram-sha-256** authentication mechanism through the Simple Authentication Security Layer (SASL) framework. The metrics RHEL System Role automates the steps to setup authentication using the **scram-sha-256** authentication mechanism. This procedure describes how to setup authentication using the metrics RHEL System Role.

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the **rhel-system-roles** package installed on the machine you want to use to run the playbook.

Procedure

1. Include the following variables in the Ansible playbook you want to setup authentication for:

```
---
  vars:
    metrics_username: your_username
    metrics_password: your_password
```

2. Run the Ansible playbook:

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

Verification steps

- Verify the **sasl** configuration:

```
# pminfo -f -h "pcp://127.0.0.1?username=your_username" disk.dev.read
Password:
disk.dev.read
inst [0 or "sda"] value 19540
```

10.6. USING THE METRICS SYSTEM ROLE TO CONFIGURE AND ENABLE METRICS COLLECTION FOR SQL SERVER

This procedure describes how to use the metrics RHEL System Role to automate the configuration and enabling of metrics collection for Microsoft SQL Server via **pcp** on your local system.

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the **rhel-system-roles** package installed on the machine you want to monitor.
- You have installed Microsoft SQL Server for Red Hat Enterprise Linux and established a 'trusted' connection to an SQL server.
- You have installed the Microsoft ODBC driver for SQL Server for Red Hat Enterprise Linux.

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 that contains the following content:

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

3. Run the Ansible playbook:

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

Verification steps

- Use the **pcp** command to verify that SQL Server PMDA agent (mssql) is loaded and running:

```
# pcp
platform: Linux rhel82-2.local 4.18.0-167.el8.x86_64 #1 SMP Sun Dec 15 01:24:23 UTC
2019 x86_64
 hardware: 2 cpus, 1 disk, 1 node, 2770MB RAM
 timezone: PDT+7
 services: pmcd pmproxy
     pmcd: Version 5.0.2-1, 12 agents, 4 clients
     pmda: root pmcd proc pmproxy xfs linux nfsclient mmv kvm mssql
           jbd2 dm
 pmlogger: primary logger: /var/log/pcp/pmlogger/rhel82-2.local/20200326.16.31
     pmie: primary engine: /var/log/pcp/pmie/rhel82-2.local/pmie.log
```

Additional resources

- For more information about using Performance Co-Pilot for Microsoft SQL Server, see this Red Hat Developers Blog post.

CHAPTER 11. 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.

11.1. 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**

11.2. 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.**

11.3. 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.
- The Ansible Core package is installed on the control machine.
- The **rhel-system-roles** package is installed on the control machine.

Procedure

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

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

11.4. DEPLOYING THE TLOG RHEL SYSTEM ROLE FOR EXCLUDING LISTS OF GROUPS OR USERS

You can use the **tlog** System Role on RHEL to support the SSSD session recording configuration options **exclude_users** and **exclude_groups**. Follow these steps to prepare and apply an Ansible playbook to configure a RHEL system to exclude users or groups from having their sessions recorded and logged in the systemd journal.

Prerequisites

- You have set SSH keys for access from the control node to the target system on which you want to configure the tlog System Role.
- The Ansible Core package is installed on the control machine.
- The **rhel-system-roles** package is installed on the control machine.

Procedure

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

```
---
- name: Deploy session recording excluding users and groups
  hosts: all
  vars:
    tlog_scope_sssd: all
    tlog_exclude_users_sssd:
      - jeff
      - james
    tlog_exclude_groups_sssd:
      - admins

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

Where,

- **tlog_scope_sssd**:
  - **all**: specifies that you want to record all users and groups.
- **tlog_exclude_users_sssd**:
  - user names: specifies the user names of the users you want to exclude from the session recording.

- **tlog_exclude_groups_sssd:**
  - **admins** specifies the group you want to exclude from the session recording.

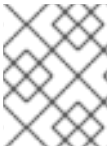
2. Optionally, verify the playbook syntax;

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

3. Run the playbook on your inventory file:

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

As a result, the playbook installs the **tlog** package on the system you specified. It also creates an **/etc/sssd/conf.d/sssd-session-recording.conf** SSSD configuration drop file that can be used by users and groups except those that you defined as excluded. SSSD parses and reads these users and groups to overlap **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.



NOTE

You are not able to record a session for users listed in the **exclude_users** list or if they are a member of a group in the **exclude_groups** list.

Verification steps

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

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

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

2. Check the file content:

```
# cat sssd-session-recording.conf
```

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

Additional resources

- See the **/usr/share/doc/rhel-system-roles/tlog/** and **/usr/share/ansible/roles/rhel-system-roles.tlog/** directories.
- The Recording a session using the deployed tlog system role in the CLI

11.5. 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.

```
# journalctl -xel _EXE=/usr/bin/tlog-rec-session
```

11.6. WATCHING A RECORDED SESSION USING THE CLI

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

Prerequisites

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

Procedure

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

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

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

```
$ /tlog-rec
```

You can see details such as:

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

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

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

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