Red Hat Enterprise Linux 7

Administration and configuration tasks using System Roles in RHEL 7.9

Applying RHEL System Roles using Red Hat Ansible Automation Platform playbooks to perform system administration tasks
Red Hat Enterprise Linux 7 Administration and configuration tasks using System Roles in RHEL 7.9

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

Applying RHEL System Roles using Red Hat Ansible Automation Platform playbooks to perform system administration tasks in RHEL 7.9
# Table of Contents

## MAKING OPEN SOURCE MORE INCLUSIVE

- PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

## CHAPTER 1. GETTING STARTED WITH RHEL SYSTEM ROLES

- 1.1. INTRODUCTION TO RHEL SYSTEM ROLES
- 1.2. RHEL SYSTEM ROLES TERMINOLOGY
- 1.3. APPLYING A ROLE
- 1.4. ADDITIONAL RESOURCES

## CHAPTER 2. INSTALLING RHEL SYSTEM ROLES

- 2.1. INSTALLING RHEL SYSTEM ROLES IN YOUR SYSTEM

## CHAPTER 3. INSTALLING AND USING COLLECTIONS

- 3.1. INTRODUCTION TO ANSIBLE COLLECTIONS
- 3.2. COLLECTIONS STRUCTURE
- 3.3. INSTALLING COLLECTIONS BY USING THE CLI
- 3.4. INSTALLING COLLECTIONS FROM AUTOMATION HUB
- 3.5. APPLYING A LOCAL LOGGING SYSTEM ROLE USING COLLECTIONS

## CHAPTER 4. USING ANSIBLE ROLES TO PERMANENTLY CONFIGURE KERNEL PARAMETERS

- 4.1. INTRODUCTION TO THE KERNEL SETTINGS ROLE
- 4.2. APPLYING SELECTED KERNEL PARAMETERS USING THE KERNEL SETTINGS ROLE

## CHAPTER 5. USING SYSTEM ROLES TO CONFIGURE NETWORK CONNECTIONS

- 5.1. CONFIGURING A STATIC ETHERNET CONNECTION USING RHEL SYSTEM ROLES WITH THE INTERFACE NAME
- 5.2. CONFIGURING A DYNAMIC ETHERNET CONNECTION USING RHEL SYSTEM ROLES WITH THE INTERFACE NAME
- 5.3. CONFIGURING VLAN TAGGING USING SYSTEM ROLES
- 5.4. CONFIGURING A NETWORK BRIDGE USING RHEL SYSTEM ROLES
- 5.5. CONFIGURING A NETWORK BOND USING RHEL SYSTEM ROLES
- 5.6. CONFIGURING A STATIC ETHERNET CONNECTION WITH 802.1X NETWORK AUTHENTICATION USING RHEL SYSTEM ROLES
- 5.7. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION USING SYSTEM ROLES
- 5.8. CONFIGURING A STATIC ROUTE USING RHEL SYSTEM ROLES
- 5.9. USING SYSTEM ROLES TO SET ETHTOOL FEATURES
- 5.10. USING SYSTEM ROLES TO CONFIGURE ETHTOOL COALESCE SETTINGS

## CHAPTER 6. CONFIGURING SELINUX USING SYSTEM ROLES

- 6.1. INTRODUCTION TO THE SELINUX SYSTEM ROLE
- 6.2. USING THE SELINUX SYSTEM ROLE TO APPLY SELINUX SETTINGS ON MULTIPLE SYSTEMS

## CHAPTER 7. USING THE LOGGING SYSTEM ROLE

- 7.1. THE LOGGING SYSTEM ROLE
- 7.2. LOGGING SYSTEM ROLE PARAMETERS
- 7.3. APPLYING A LOCAL LOGGING SYSTEM ROLE
- 7.4. FILTERING LOGS IN A LOCAL LOGGING SYSTEM ROLE
- 7.5. APPLYING A REMOTE LOGGING SOLUTION USING THE LOGGING SYSTEM ROLE
- 7.6. ADDITIONAL RESOURCES
- 7.7. USING THE LOGGING SYSTEM ROLES WITH RELP
  - 7.7.1. Configuring client logging with RELP
  - 7.7.2. Configuring server logging with RELP
- 7.8. USING THE LOGGING SYSTEM ROLE WITH TLS
7.8.1. Configuring client logging with TLS
7.8.2. Configuring server logging with TLS

CHAPTER 8. CONFIGURING SECURE COMMUNICATION WITH THE SSH SYSTEM ROLES
8.1. SSHD SYSTEM ROLE VARIABLES
8.2. Configuring OpenSSH servers using the SSHD system role
8.3. SSH system role variables
8.4. Configuring OpenSSH clients using the SSH system role

CHAPTER 9. SETTING A CUSTOM CRYPTO POLICY ACROSS SYSTEMS
9.1. Crypto policies system role variables and facts
9.2. Setting a custom crypto policy using the crypto policies system role
9.3. Additional resources

CHAPTER 10. USING THE NBDE_CLIENT AND NBDE_SERVER SYSTEM ROLES
10.1. Introduction to the NBDE_CLIENT and NBDE_SERVER system roles
10.2. Using the NBDE_SERVER system role for setting up multiple Tang servers
10.3. Using the NBDE_CLIENT system role for setting up multiple Clevis clients

CHAPTER 11. REQUESTING CERTIFICATES USING RHEL SYSTEM ROLES
11.1. The Certificate system role
11.2. Requesting a new self-signed certificate using the Certificate system role
11.3. Requesting a new certificate from IDM CA using the Certificate system role
11.4. Specifying commands to run before or after certificate issuance using the Certificate system role

CHAPTER 12. CONFIGURING KDUMP USING RHEL SYSTEM ROLES
12.1. The KDUMP RHEL system role
12.2. KDUMP role parameters
12.3. Configuring KDUMP using RHEL system roles

CHAPTER 13. MANAGING LOCAL STORAGE USING RHEL SYSTEM ROLES
13.1. Introduction to the Storage role
13.2. Parameters that identify a storage device in the Storage system role
13.3. Example Ansible playbook to create an XFS file system on a block device
13.4. Example Ansible playbook to persistently mount a file system
13.5. Example Ansible playbook to manage logical volumes
13.6. Example Ansible playbook to enable online block discard
13.7. Example Ansible playbook to create and mount an EXT4 file system
13.8. Example Ansible playbook to create and mount an EXT3 file system
13.9. Example Ansible playbook to resize an existing EXT4 or EXT3 file system using the Storage RHEL system role
13.10. Example Ansible playbook to resize an existing file system on LVM using the Storage RHEL system role
13.11. Example Ansible playbook to create a swap partition using the Storage RHEL system role
13.12. Configuring a RAID volume using the Storage system role
13.13. Configuring an LVM pool with RAID using the Storage system role
13.14. Creating a LUKS encrypted volume using the Storage role
13.15. Additional resources

CHAPTER 14. CONFIGURING TIME SYNCHRONIZATION USING RHEL SYSTEM ROLES
14.1. The TIMESYNC system role
14.2. Applying the TIMESYNC system role for a single pool of servers
14.3. TIMESYNC system roles variables
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.

- 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 feedback via Bugzilla, create a new ticket:
  1. Go to the [Bugzilla](https://bugzilla.redhat.com) 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 {RHEL7.9}, the interface currently consists of the following roles:

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

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-VERSION` documentation
- Introduction to the SELinux system role
- Introduction to the storage role

1.2. RHEL SYSTEM ROLES TERMINOLOGY

You can find the following terms across this documentation:
System Roles terminology

Ansible playbook

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

Control node

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

Inventory

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

Managed nodes

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

1.3. APPLYING A ROLE

The following procedure describes how to apply a particular role.

Prerequisites

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

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

- You need the `ansible` package to run playbooks that use RHEL System Roles. Ensure that the Ansible Engine repository is enabled, and the `ansible` package is installed on the system that you want to use as a control node.

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

    1. Enable the RHEL Ansible Engine repository:

       ```bash
       # subscription-manager refresh
       # sudo subscription-manager repos --enable rhel-7-server-ansible-2.9-rpms
       ```

    2. Install Ansible Engine:

       ```bash
       # yum install ansible
       ```

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

- 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.

Inventories 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:

   ```ini
   [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:

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

   **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-VERSION/SUBSYSTEM/
   ```

   For the rhel-7.9 rhel-system-roles package, the VERSION value is 1.0.1.

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

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:

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

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

  ```bash
  # 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:

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

Playbook execution command:

```bash
# 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
- Managing local storage using RHEL System Roles
- Deploying the same SELinux configuration on multiple systems using RHEL System Roles

[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 RHEL System Roles, install the required packages.

Prerequisites

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

Procedure

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

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

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

   a. Enable the RHEL Ansible Engine repository:

      ```
      # subscription-manager refresh
      # sudo subscription-manager repos --enable rhel-7-server-ansible-2.9-rpms
      ```

   b. Install Ansible Engine:

      ```
      # yum install ansible
      ```

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

Additional resources

- For RHEL System Roles overview, see the Red Hat Enterprise Linux (RHEL) System Roles
- For more detailed information on using the ansible-playbook command, see the ansible-playbook man page.
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 the Automation Hub, through the browser, or by using the command line.

Prerequisites

- Red Hat Ansible Engine version 2.9 and later is installed.
- 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/doc/rhel-system-roles-VERSION/collection/README.md` and `/usr/share/doc/rhel-system-roles-VERSION/collection/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/tests_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

- See 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
Prerequisites

- Red Hat Ansible Engine version 2.9 or later is installed.
- 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/tests_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

- See the `ansible-playbook` man page.

3.5. APPLYING A LOCAL LOGGING SYSTEM ROLE USING COLLECTIONS

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

Prerequisites
A Collection format of `rhel-system-roles` is installed either from a `rpm` package or from the Automation Hub.

**Procedure**

1. Create a playbook that defines the required role:
   a. Create a new YAML file and open it in a text editor, for example:
      ```bash
      # vi logging-playbook.yml
      ```
   b. Insert the following content into the YAML file:
      ```yaml
      ---
      - name: Deploying basics input and implicit files output
        hosts: all
        roles:
        - redhat.rhel_system_roles.logging
        vars:
          logging_inputs:
            - name: system_input
              type: basics
          logging_outputs:
            - name: files_output
              type: files
          logging_flows:
            - name: flow1
              inputs: [system_input]
              outputs: [files_output]
      ```

2. Execute the playbook on a specific inventory:
   ```bash
   # ansible-playbook -i inventory-file logging-playbook.yml
   ```
   Where:
   - `inventory-file` is the name of your inventory file.
   - `logging-playbook.yml` is the playbook you use.

**Verification steps**

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

2. Verify that the system sends messages to the log:
   a. Send a test message:
      ```bash
      # logger test
      ```
b. View the /var/log/messages log, for example:

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

The **hostname** is the hostname of the client system. The log displays the user name of the user that entered the logger command, in this case, **root**.
CHAPTER 4. USING ANSIBLE ROLES TO PERMANENTLY CONFIGURE KERNEL PARAMETERS

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

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

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

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 Engine to configure according to the playbook.

With the kernel_settings role you can configure:

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

Additional resources

- README.md files in the /usr/share/doc/rhel-system-roles-VERSION/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**

- Your Red Hat Ansible Engine subscription is attached to the system, also called *control machine*, from which you want to run the *kernel_settings* role. See the [How do I download and install Red Hat Ansible Engine](http://example.com) article for more information.

- Ansible Engine repository is enabled on the control machine.

- Ansible Engine is installed on the control machine.

**NOTE**

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

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

- An inventory of managed hosts is present on the control machine and Ansible Engine is able to connect to them.

**Procedure**

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

   ```
   # cat /home/jdoe/<ansible_project_name>/inventory
   [testingservers]
   pdoe@192.168.122.98
   fdoe@192.168.122.226
   
   [db-servers]
   db1.example.com
   db2.example.com
   
   [webservers]
   web1.example.com
   web2.example.com
   192.0.2.42
   
   The file defines the [*testing.servers*] group and other groups. It allows you to run Ansible Engine more effectively against a specific collection of systems.
   ```

2. Create a configuration file to set defaults and privilege escalation for Ansible Engine operations.
   a. Create a new YAML file and open it in a text editor, for example:

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

   b. Insert the following content into the file:

   ```
   [defaults]
   inventory = ./inventory
   ```
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:

   ```yaml
   # 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:

   ```yaml
   ---
   - 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:
       - rhel-system-roles.kernel_settings
   ```

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

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

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

   **NOTE**

   You can modify the kernel parameters and their values in the playbook to fit your needs.
4. Optionally, verify that the syntax in your play is correct.

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

playbook: kernel-roles.yml

This example shows the successful verification of a playbook.

5. Execute your playbook.

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

BECOME password:

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

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

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

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

Additional resources

- Getting started with RHEL System Roles
- README.md files in the /usr/share/doc/rhel-system-roles-VERSION/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 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:

   ```yaml
   ---
   - name: Configure an Ethernet connection with static IP
     hosts: node.example.com
     become: true
     tasks:
       - include_role:
           name: rhel-system-roles.network
   ```
vars:
    network_connections:
    - name: enp7s0
      interface_name: enp7s0
      type: ethernet
      autoconnect: yes
      ip:
        address:
        - 192.0.2.1/24
        - 2001:db8:1::1/64
        gateway4: 192.0.2.254
        gateway6: 2001:db8:1::ffee
        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 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 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:

   ```yaml
   ---
   - name: Configure an Ethernet connection with dynamic IP
     hosts: node.example.com
     become: true
     tasks:
     - include_role:
         name: rhel-system-roles.network
     vars:
         network_connections:
         - name: enp7s0
           interface_name: enp7s0
           type: ethernet
           autoconnect: yes
           ip:
             dhcp4: yes
             auto6: yes
           state: up
   ```

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

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

- To use the VLAN as a port in other connections, such as a bond, omit the `ip` attribute, and set the IP configuration in the parent configuration.

- To use team, bridge, or bond devices in the VLAN, adapt the `interface_name` and `type` attributes of the ports you use in the VLAN.

**Prerequisites**

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

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

**Procedure**

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

```shell
node.example.com
```

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

5.4. CONFIGURING A NETWORK BRIDGE USING RHEL SYSTEM ROLES

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

NOTE

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

Prerequisites

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

• Two or more physical or virtual network devices are installed on the server.

Procedure

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

```
node.example.com
```

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

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

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

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

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

5.5. CONFIGURING A NETWORK BOND USING RHEL SYSTEM ROLES

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

NOTE

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

Prerequisites

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

Procedure

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

   ```
   node.example.com
   ```

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

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

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

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

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

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

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

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

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

Run this procedure on the Ansible control node.

Prerequisites

- The ansible and rhel-system-roles packages are installed on the control node.
- If you use a different remote user than root when you run the playbook, you must have appropriate sudo permissions on the managed node.
- The network supports 802.1X network authentication.
- The managed node uses NetworkManager.
- The following files required for TLS authentication exist on the control node:
  - The client key 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:
2. Create the `~/.enable-802.1x.yml` playbook with the following content:

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

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

### 5.7. 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::ffe
- An IPv4 DNS server - 198.51.100.200
- An IPv6 DNS server - 2001:db8:1::ffbb
- A DNS search domain - example.com

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

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

Procedure

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

   node.example.com

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

   ```yaml
   ---
   - name: Configure an Ethernet connection with static IP and default gateway
     hosts: node.example.com
     become: true
     tasks:
     - include_role:
       name: rhel-system-roles.network
       vars:
         network_connections:
           - name: enp1s0
             type: ethernet
             autoconnect: yes
             ip:
               address:
                 - 198.51.100.20/24
                 - 2001:db8:1::1/64
               gateway4: 198.51.100.254
               gateway6: 2001:db8:1::fffe
             dns:
               - 198.51.100.200
               - 2001:db8:1::ffbb
             dns_search:
               - example.com
             state: up
   
   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.
5.8. CONFIGURING A STATIC ROUTE USING RHEL SYSTEM ROLES

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

IMPORTANT

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

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

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

Prerequisites

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

Procedure

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

   ```
   node.example.com
   ```
2. Create the ~/add-static-routes.yml playbook with the following content:

```yaml
---
- name: Configure an Ethernet connection with static IP and additional routes
  hosts: node.example.com
  become: true
  tasks:
    - include_role:
      name: rhel-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::ffe
        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:
     ```bash
     # ansible-playbook -u root ~/add-static-routes.yml
     ```

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

5.9. USING SYSTEM ROLES TO SET ETHTOOL FEATURES

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

IMPORTANT

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

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

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

Prerequisites

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

**Procedure**

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

```
node.example.com
```

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

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

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

### 5.10. 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::ffe
- 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` 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:

   ```yaml
   ---
   - name: Configure an Ethernet connection with ethtool coalesce settings
     hosts: node.example.com
     become: true
     tasks:
     - include_role:
       name: rhel-system-roles.network
     vars:
       network_connections:
       - name: enp1s0
         type: ethernet
         autoconnect: yes
         ip:
           address:
           - 198.51.100.20/24
           - 2001:db8:1::1/64
           gateway4: 198.51.100.254
           gateway6: 2001:db8:1::fffe
           dns:
           - 198.51.100.200
           - 2001:db8:1::ffff
           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 SELINUX USING SYSTEM ROLES

6.1. INTRODUCTION TO THE SELINUX SYSTEM ROLE

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

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

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

Table 6.1. SELinux system role variables

<table>
<thead>
<tr>
<th>Role variable</th>
<th>Description</th>
<th>CLI alternative</th>
</tr>
</thead>
<tbody>
<tr>
<td>selinux_policy</td>
<td>Chooses a policy protecting targeted processes or Multi Level Security protection.</td>
<td>SELINUXTYPE in /etc/selinux/config</td>
</tr>
<tr>
<td>selinux_state</td>
<td>Switches SELinux modes. See ansible-doc selinux</td>
<td>setenforce and SELINUX in /etc/selinux/config</td>
</tr>
<tr>
<td>selinux_booleans</td>
<td>Enables and disables SELinux booleans. See ansible-doc seboolean.</td>
<td>setsebool</td>
</tr>
<tr>
<td>selinux_fcontexts</td>
<td>Adds or removes a SELinux file context mapping. See ansible-doc sefcontext.</td>
<td>semanage fcontext</td>
</tr>
<tr>
<td>selinux_restore_dirs</td>
<td>Restores SELinux labels in the file-system tree.</td>
<td>restorecon -R</td>
</tr>
<tr>
<td>selinux_ports</td>
<td>Sets SELinux labels on ports. See ansible-doc seport.</td>
<td>semanage port</td>
</tr>
<tr>
<td>selinux_logins</td>
<td>Sets users to SELinux user mapping. See ansible-doc sellogin.</td>
<td>semanage login</td>
</tr>
<tr>
<td>selinux_modules</td>
<td>Installs, enables, disables, or removes SELinux modules.</td>
<td>semodule</td>
</tr>
</tbody>
</table>
The /usr/share/ansible/roles/rhel-system-roles.selinux/selinux-playbook.yml example Playbook installed by the rhel-system-roles package demonstrates how to set the targeted policy in enforcing mode. The playbook also applies several local policy modifications and restores file contexts in the /tmp/test_dir/ directory.

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

Additional resources

- Introduction to RHEL System Roles

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

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

**Prerequisites**

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

**Procedure**

1. Enable the RHEL Ansible repository, for example:
   ```bash
   # sudo subscription-manager repos --enable rhel-7-server-ansible-2.9-rpms
   ```
2. Install Ansible Engine:
   ```bash
   # yum install ansible
   ```
3. Install RHEL system roles:
   ```bash
   # yum install rhel-system-roles
   ```
4. Prepare your playbook. You can either start from the scratch or modify the example playbook installed as a part of the rhel-system-roles package:
   ```bash
   # cp /usr/share/ansible/roles/rhel-system-roles.selinux/selinux-playbook.yml my-selinux-playbook.yml
   # vi my-selinux-playbook.yml
   ```
5. Change the content of the playbook to fit your scenario. For example, the following part ensures that the system installs and enables the selinux-local-1.pp SELinux module:
   ```yml
   selinux_modules:
   - { path: "selinux-local-1.pp", priority: "400" }
   ```
6. Save the changes, and exit the text editor.
7. Run your playbook on the host1, host2, and host3 systems:
```
# ansible-playbook -i host1,host2,host3 my-selinux-playbook.yml
```

**Additional resources**

- For more information, install the `rhel-system-roles` package, and see the `/usr/share/doc/rhel-system-roles-VERSION/selinux/` directory.
CHAPTER 7. USING THE LOGGING SYSTEM ROLE

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

7.1. THE LOGGING SYSTEM ROLE

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

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

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

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

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

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

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

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

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

7.2. LOGGING SYSTEM ROLE PARAMETERS

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

NOTE

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

- **logging_inputs**: List of inputs for the logging solution.
  - **name**: Unique name of the input. Used in the logging_flows: inputs list and a part of the generated config file name.
- **type**: Type of the input element. The type specifies a task type which corresponds to a directory name in `roles/rsyslog/(tasks,vars)/inputs/`.
  - **basics**: Inputs configuring inputs from `systemd` journal or `unix` socket.
    - **kernel_message**: Load `imklog` if set to `true`. Default to `false`.
    - **use_imuxsock**: Use `imuxsock` instead of `imjournal`. Default to `false`.
    - **ratelimit_burst**: Maximum number of messages that can be emitted within `ratelimit_interval`. Default to `20000` if `use_imuxsock` is false. Default to `200` if `use_imuxsock` is true.
    - **ratelimit_interval**: Interval to evaluate `ratelimit_burst`. Default to `600` seconds if `use_imuxsock` is false. Default to `0` if `use_imuxsock` is true. `0` indicates rate limiting is turned off.
    - **persist_state_interval**: Journal state is persisted to every `value` message. Default to `10`. Effective only when `use_imuxsock` is false.
  - **files**: Inputs configuring inputs from local files.
  - **remote**: Inputs configuring inputs from the other logging system over network.
    - **state**: State of the configuration file. `present` or `absent`. Default to `present`.
    - **logging_outputs**: List of outputs for the logging solution.
      - **files**: Outputs configuring outputs to local files.
      - **forwards**: Outputs configuring outputs to another logging system.
      - **remote_files**: Outputs configuring outputs from another logging system to local files.
    - **logging_flows**: List of flows that define relationships between `logging_inputs` and `logging_outputs`. The `logging_flows` variable has the following keys:
      - **name**: Unique name of the flow
      - **inputs**: List of `logging_inputs` name values
      - **outputs**: List of `logging_outputs` name values.

### Additional resources
- Documentation installed with the `rhel-system-roles` package in `/usr/share/ansible/roles/rhel-system-roles.logging/README.md`

#### 7.3. APPLYING A LOCAL LOGGING SYSTEM ROLE

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

### Prerequisites

- Access and permissions to one or more **managed nodes**, which are systems you want to configure with the Crypto Policies System Role.
• Access and permissions to a **control node**, which is a system from which Red Hat Ansible Engine configures other systems.

On the control node:

- Red Hat Ansible Engine is installed
- The **rhel-system-roles** package is installed
- An inventory file which lists the managed nodes.

**Procedure**

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

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

   b. Insert the following content:

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

2. Execute the playbook on a specific inventory:

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

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

**Verification**

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

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

2. Verify that the system sends messages to the log:
   
a. Send a test message:
# logger test

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

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

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

### 7.4. FILTERING LOGS IN A LOCAL LOGGING SYSTEM ROLE

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

#### Prerequisites

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

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

  On the control node:

  - Red Hat Ansible Engine is installed
  - The `rhel-system-roles` package is installed
  - An inventory file which lists the managed nodes.

#### Procedure

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

```yaml
---
- name: Deploying files input and configured files output
  hosts: all
  roles:
  - rhel-system-roles.logging
  vars:
    logging_inputs:
      - name: files_input0
        type: files
        input_log_path: /var/log/containerA/*.log
      - name: files_input1
        type: files
        input_log_path: /var/log/containerB/*.log
    logging_outputs:
      - name: files_output0
        type: files
        property: msg
        property_op: contains
        property_value: error
        path: /var/log/errors.log
      - name: files_output1
```

Red Hat Enterprise Linux 7 Administration and configuration tasks using System Roles in RHEL 7.9
Using this configuration, all messages that contain the `error` string are logged in /var/log/errors.log, and all other messages are logged in /var/log/others.log.

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

You can modify the variables according to your preferences.

2. Optional: Verify playbook syntax.

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

3. Run the playbook on your inventory file:

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

Verification

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

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

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

   a. Send a test message:

      # logger error

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

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

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

Additional resources

- Documentation installed with the `rhel-system-roles` package in
  /usr/share/ansible/roles/rhel-system-roles.logging/README.md
7.5. APPLYING A REMOTE LOGGING SOLUTION USING THE LOGGING SYSTEM ROLE

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

Prerequisites

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

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

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

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

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

Procedure

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

      # vi logging-playbook.yml

   b. Insert the following content into the file:

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

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

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

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

**WARNING**
The logging solution works only with the ports defined in the SELinux policy of the server or client system and open in the firewall. The default SELinux policy includes ports 601, 514, 6514, 10514, and 20514. To use a different port, modify the SELinux policy on the client and server systems. Configuring the firewall through system roles is not yet supported.
2. Create an inventory file that lists your servers and clients:
   a. Create a new file and open it in a text editor, for example:

   ```
   # vi inventory.ini
   ```

   b. Insert the following content into the inventory file:

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

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

3. Execute the playbook on your inventory.

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

   Where:
   - `inventory.ini` is the inventory file.
   - `logging-playbook.yml` is the playbook you created.

Verification

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

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

2. Verify that the client system sends messages to the server:
   a. On the client system, send a test message:

   ```
   # logger test
   ```

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

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

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

Additional resources
7.6. ADDITIONAL RESOURCES

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

7.7. USING THE LOGGING SYSTEM ROLES WITH RELP

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

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

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

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

7.7.1. Configuring client logging with RELP

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

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

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

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

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

The playbooks uses following settings:

- **target**: This is a required parameter that specifies the host name where the remote logging system is running.

- **port**: Port number the remote logging system is listening.

- **tls**: Ensures secure transfer of logs over the network. If you do not want a secure wrapper you can set the `tls` variable to `false`. By default `tls` parameter is set to `true` while working with RELP and requires key/certificates and triplets `{ca_cert, cert, private_key}` and/or `{ca_cert_src, cert_src, private_key_src}`.
  - If `{ca_cert_src, cert_src, private_key_src}` triplet is set, the default locations `/etc/pki/tls/certs` and `/etc/pki/tls/private` are used as the destination on the managed node to transfer files from control node. In this case, the file names are identical to the original ones in the triplet.
  - If `{ca_cert, cert, private_key}` triplet is set, files are expected to be on the default path before the logging configuration.
  - If both the triplets are set, files are transferred from local path from control node to specific path of the managed node.

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

- **cert**: Represents the path to cert. Default is `/etc/pki/tls/certs/server-cert.pem` and the file name is set by the user.
• **private_key**: Represents the path to private key. Default path is `/etc/pki/tls/private/server-key.pem` and the file name is set by the user.

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

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

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

• **pki_authmode**: Accepts the authentication mode as `name` or `fingerprint`.

• **permitted_servers**: List of servers that will be allowed by the logging client to connect and send logs over TLS.

• **inputs**: List of logging input dictionary.

• **outputs**: List of logging output dictionary.

2. Optional: Verify playbook syntax.

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

3. Run the playbook:

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

### 7.7.2. Configuring server logging with RELP

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

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

**Prerequisites**

- You have permissions to run playbooks on managed nodes on which you want to configure RELP.

- The managed nodes are listed in the inventory file on the control node.

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

**Procedure**

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

   ```yaml
   ---
   - name: Deploying remote input and remote_files output
     hosts: server
     roles:
   ```
- rhel-system-roles.logging

vars:
  logging_inputs:
    - name: relp_server
      type: relp
      port: 20514
      tls: true
      ca_cert: /etc/pki/tls/certs/ca.pem
      cert: /etc/pki/tls/certs/server-cert.pem
      private_key: /etc/pki/tls/private/server-key.pem
      pki_authmode: name
      permitted_clients:
        - '*example.client.com'
  logging_outputs:
    - name: _remote_files_output_
      type: _remote_files_
  logging_flows:
    - name: _example_flow_
      inputs: _relp_server_
      outputs: _remote_files_output_

The playbooks uses following settings:

- **port**: Port number the remote logging system is listening.

- **tls**: Ensures secure transfer of logs over the network. If you do not want a secure wrapper you can set the tls variable to false. By default tls parameter is set to true while working with RELP and requires key/certificates and triplets {ca_cert, cert, private_key} and/or {ca_cert_src, cert_src, private_key_src}.
  - If {ca_cert_src, cert_src, private_key_src} triplet is set, the default locations /etc/pki/tls/certs and /etc/pki/tls/private are used as the destination on the managed node to transfer files from control node. In this case, the file names are identical to the original ones in the triplet.
  - If {ca_cert, cert, private_key} triplet is set, files are expected to be on the default path before the logging configuration.
  - If both the triplets are set, files are transfered from local path from control node to specific path of the managed node.

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

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

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

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

- **cert_src**: Repsents the local cert file path which is copied to the target host. If cert is specified, it is copied to the location.
7.8. USING THE LOGGING SYSTEM ROLE WITH TLS

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

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

7.8.1. Configuring client logging with TLS

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

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

Prerequisites

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

Procedure

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

```yaml
---
- name: Deploying files input and forwards output with certs
  hosts: clients
  roles:
    - rhel-system-roles.logging
```
The playbook uses the following parameters:

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

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

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

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

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

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

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

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

2. Verify playbook syntax:
3. Run the playbook on your inventory file:

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

### 7.8.2. Configuring server logging with TLS

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

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

**Prerequisites**

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

**Procedure**

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

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

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

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

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

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

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

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

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

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

2. Verify playbook syntax:

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

3. Run the playbook on your inventory file:

```
# ansible-playbook -i inventory_file playbook.yml
```
CHAPTER 8. CONFIGURING SECURE COMMUNICATION WITH THE SSH SYSTEM ROLES

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

8.1. 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, whereas `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

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:

```python
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:

```python
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 8, 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.

8.2. 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.
- Access and permissions to a control node, which is a system from which Red Hat Ansible Engine configures other systems.
  - On the control node:
    - Red Hat Ansible Engine is installed.
    - The rhel-system-roles package is installed.
    - An inventory file which lists the managed nodes.

Procedure

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

   ```
   # cp /usr/share/doc/rhel-system-roles-VERSION/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 System Role variables.

3. Optional: Verify playbook syntax.

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

4. Run the playbook on your inventory file:

```bash
# 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:

```bash
$ 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:

```bash
$ 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

- ansible-playbook(1) man page.

8.3. 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.

### 8.4. CONFIGURING OPENSSSH 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.

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

  On the control node:

  - Red Hat Ansible Engine is installed.
  - The `rhel-system-roles` package is installed.
  - An inventory file which lists the managed nodes.

**Procedure**

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

   ```yaml
   ---
   - 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
   ```
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 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 9. SETTING A CUSTOM CRYPTO 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 Red Hat Ansible Automation Platform.

9.1. 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 level the system role applies to the managed nodes. For details about the different crypto policy levels, 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 policy levels available on the system.

crypto_policies_available_modules
Lists all available subpolicy modules available on the system.

Additional resources

* Creating and setting a custom system-wide cryptographic policy

9.2. SETTING A CUSTOM CRYPTO POLICY USING THE CRYPTO POLICIES SYSTEM ROLE

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

Prerequisites
Access and permissions to one or more managed nodes, which are systems you want to configure with the Crypto Policies System Role.

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

On the control node:
- Red Hat Ansible Engine is installed
- The `rhel-system-roles` package is installed
- An inventory file which lists the managed nodes.

Procedure

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

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

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

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

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

2. Optional: Verify playbook syntax.

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

3. Run the playbook on your inventory file:

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

Verification

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

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

### 9.3. 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 10. USING THE NBDE_CLIENT AND NBDE_SERVER SYSTEM ROLES

10.1. INTRODUCTION TO THE NBDE_CLIENT AND NBDE_SERVER SYSTEM ROLES

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

RHEL 7.9 introduced Ansible roles for automated deployments of Policy-Based Decryption (PBD) solutions using Clevis and Tang. The rhel-system-roles package contains these system roles, the related examples, and also the reference documentation.

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

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 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.md files in the /usr/share/doc/rhel-system-roles-VERSION/nbde_client/ and /usr/share/doc/rhel-system-roles-VERSION/nbde_server/ directories.

- For example system-roles playbooks, install the rhel-system-roles package, and see the /usr/share/doc/rhel-system-roles-VERSION/nbde_server/example-*-playbook.yml and /usr/share/doc/rhel-system-roles-VERSION/nbde_client/example-*-playbook.yml directories.

- For more information on RHEL System Roles, see Introduction to RHEL System Roles
### 10.2. USING THE NBDE_SERVER SYSTEM ROLE FOR SETTING UP MULTIPLE TANG SERVERS

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

**Prerequisites**

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

**Procedure**

1. Enable the RHEL Ansible repository, for example:
   ```bash
   # sudo subscription-manager repos --enable rhel-7-server-ansible-2.9-rpms
   ```

2. Install Ansible Engine:
   ```bash
   # yum install ansible
   ```

3. Install RHEL system roles:
   ```bash
   # yum install rhel-system-roles
   ```

4. Prepare your playbook containing settings for Tang servers. You can either start from the scratch, or use one of the example playbooks from the `/usr/share/doc/rhel-system-roles-VERSION/nbde_server/` directory.
   ```bash
   # cp /usr/share/doc/rhel-system-roles-VERSION/nbde_server/example-simple_deploy-playbook.yml ./my-tang-playbook.yml
   ```

5. Edit the playbook in a text editor of your choice, for example:
   ```bash
   # vi my-tang-playbook.yml
   ```

6. Add the required parameters. The following example playbook ensures deploying of your Tang server and a key rotation:
   ```yaml
   ---
   - hosts: all
     vars:
       nbde_server_rotate_keys: yes
     roles:
       - rhel-system-roles.nbde_server
   ```

7. Apply the finished playbook:
   ```bash
   # ansible-playbook -i host1,host2,host3 my-tang-playbook.yml
   ```

**Additional resources**
10.3. USING THE NBDE_CLIENT SYSTEM ROLE FOR SETTING UP MULTIPLE CLEVIS CLIENTS

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

NOTE

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

Prerequisites

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

Procedure

1. Enable the RHEL Ansible repository, for example:

   # sudo subscription-manager repos --enable rhel-7-server-ansible-2.9-rpms

2. Install Ansible Engine:

   # yum install ansible

3. Install RHEL system roles:

   # yum install rhel-system-roles

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

   # cp /usr/share/doc/rhel-system-roles-VERSION/nbde_client/example-high_availability-playbook.yml ./my-clevis-playbook.yml

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

   # vi my-clevis-playbook.yml

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

   ---
   - hosts: all
7. Apply the finished playbook:

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

Additional resources

- For details about the parameters and additional information about the \texttt{nbde\_client} role, install the \texttt{rhel-system-roles} package, and see the \texttt{/usr/share/doc/rhel-system-roles-\texttt{VERSION}/nbde\_client/} directory.
CHAPTER 11. REQUESTING CERTIFICATES USING RHEL SYSTEM ROLES

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

This chapter covers the following topics:

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

11.1. THE CERTIFICATE SYSTEM ROLE

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

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

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

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

Additional resources

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

- For details about RHEL System Roles and how to apply them, see Getting started with RHEL System Roles.

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

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

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

NOTE

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

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

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

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

### Procedure

1. **Optional:** Create an inventory file, for example `inventory.file`:
   ```bash
   $ 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`:
   ```yaml
   - 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:

   ```bash
   $ 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.

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

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

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

**NOTE**

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

**Prerequisites**

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

**NOTE**

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

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

For details about RHEL System Roles and how to apply them, see [Getting started with RHEL System Roles](#).

**Procedure**

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

   ```bash
   $ touch inventory.file
   ```

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

```yaml
---
- 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.

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

With the Certificate System Role, you can use Red Hat Ansible Engine to execute a command before and after a certificate is issued or renewed.
In the following example, the administrator ensures stopping the httpd service before a self-signed certificate for www.example.com is issued or renewed, and restarting it afterwards.

NOTE

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

Prerequisites

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

NOTE

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

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

Procedure

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

   ```bash
   $ touch inventory.file
   ```

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

   ```ini
   [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:

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

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

4. Save the file.

5. Run the playbook:

   ```bash
   $ 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 12. CONFIGURING KDUMP USING RHEL SYSTEM ROLES

To manage kdump using Ansible, you can use the `kdump` role, which is one of the RHEL System Roles available in RHEL 7.9.

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

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

12.1. THE KDUMP RHEL SYSTEM ROLE

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

12.2. KDUMP ROLE PARAMETERS

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

<table>
<thead>
<tr>
<th>Role Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>kdump_path</code></td>
<td>The path to which <code>vmcore</code> is written. If <code>kdump_target</code> is not null, path is relative to that dump target. Otherwise, it must be an absolute path in the root file system.</td>
</tr>
</tbody>
</table>

Additional resources

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

12.3. CONFIGURING KDUMP USING RHEL SYSTEM ROLES

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

WARNING

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

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

**NOTE**

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

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

- You have an inventory file which lists the systems on which you want to deploy `kdump`.

Procedure

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

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

2. Optional: Verify playbook syntax.

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

3. Run the playbook on your inventory file:

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

Additional resources

- For a detailed reference on `kdump` role variables, see the README.md files in the `/usr/share/doc/rhel-system-roles-VERSION/kdump` directory.

- See [Applying a role](#).

- Documentation installed with the `rhel-system-roles` package `/usr/share/ansible/roles/rhel-system-roles.kdump/README.md`
CHAPTER 13. MANAGING LOCAL STORAGE USING RHEL SYSTEM ROLES

To manage LVM and local file systems (FS) using Ansible, you can use the storage role, which is one of the RHEL System Roles available in RHEL 7.9.

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

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

13.1. INTRODUCTION TO THE STORAGE ROLE

The storage role can manage:

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

With the storage role you can perform the following tasks:

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

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

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

storage_volumes

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

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

13.3. EXAMPLE ANSIBLE PLAYBOOK TO CREATE AN XFS FILE SYSTEM ON A BLOCK DEVICE

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

WARNING

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

Example 13.1. A playbook that creates XFS on /dev/sdb

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

- The volume name (barefs in the example) is currently arbitrary. The storage role
  identifies the volume by the disk device listed under the disks: attribute.

- You can omit the fs_type: xfs line because XFS is the default file system in RHEL 7.9.

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

Additional resources

- For details about the parameters used in the storage system role, see the
  /usr/share/ansible/roles/rhel-system-roles.storage/README.md file.
13.4. EXAMPLE ANSIBLE PLAYBOOK TO PERSISTENTLY MOUNT A FILE SYSTEM

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

Example 13.2. A playbook that mounts a file system on /dev/sdb to /mnt/data

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

- This playbook adds the file system to the /etc/fstab file, and mounts the file system immediately.

- If the file system on the /dev/sdb device or the mount point directory do not exist, the playbook creates them.
```

Additional resources

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

13.5. EXAMPLE ANSIBLE PLAYBOOK TO MANAGE LOGICAL VOLUMES

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

Example 13.3. A playbook that creates a mylv logical volume in the myvg volume group

```yaml
- hosts: all
  vars:
    storage_pools:
      - name: myvg
        disks:
          - sda
          - sdb
          - sdc
    volumes:
      - name: mylv
        size: 2G
        fs_type: ext4
```
mount_point: /mnt
roles:
  - rhel-system-roles.storage

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

Additional resources

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

13.6. EXAMPLE ANSIBLE PLAYBOOK TO ENABLE ONLINE BLOCK DISCARD

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

Example 13.4. A playbook that enables online block discard on /mnt/data/

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

Additional resources

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

### 13.7. EXAMPLE ANSIBLE PLAYBOOK TO CREATE AND MOUNT AN EXT4 FILE SYSTEM

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

**Example 13.5. A playbook that creates Ext4 on `/dev/sdb` and mounts it at `/mnt/data`**

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

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

**Additional resources**

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

### 13.8. EXAMPLE ANSIBLE PLAYBOOK TO CREATE AND MOUNT AN EXT3 FILE SYSTEM

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

**Example 13.6. A playbook that creates Ext3 on `/dev/sdb` and mounts it at `/mnt/data`**

```yaml
---
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
```

Red Hat Enterprise Linux 7 Administration and configuration tasks using System Roles in RHEL 7.9
The playbook creates the file system on the /dev/sdb disk.

The playbook persistently mounts the file system at the /mnt/data directory.

The label of the file system is label-name.

Additional resources

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

13.9. EXAMPLE ANSIBLE PLAYBOOK TO RESIZE AN EXISTING EXT4 OR EXT3 FILE SYSTEM USING THE STORAGE RHEL SYSTEM ROLE

This section provides an example Ansible playbook. This playbook applies the storage role to resize an existing Ext4 or Ext3 file system on a block device.

Example 13.8. A playbook that resizes ext4 on /dev/sdb

---
- name: Create a disk device mounted on /opt/barefs
- hosts: all
  vars:
    storage_volumes:
    - name: barefs
      type: disk
      disks:
      - /dev/sdb
      size: 12 GiB
      fs_type: ext4
      mount_point: /opt/barefs
      roles:
      - rhel-system-roles.storage

- If the volume in the previous example already exists, to resize the volume, you need to run the same playbook, just with a different value for the parameter size. For example:
type: disk
disks:
- /dev/sdb

size: 10 GiB
fs_type: ext4
mount_point: /opt/barefs
roles:
- rhel-system-roles.storage

- The volume name (barefs in the example) is currently arbitrary. The storage role identifies the volume by the disk device listed under the disks: attribute.

**NOTE**

Using the Resizing action in other file systems can destroy the data on the device you are working on.

Additional resources

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

### 13.10. EXAMPLE ANSIBLE PLAYBOOK TO RESIZE AN EXISTING FILE SYSTEM ON LVM USING THE STORAGE RHEL SYSTEM ROLE

This section provides an example Ansible playbook. This playbook applies the storage RHEL System Role to resize an LVM logical volume with a file system.

**WARNING**

Using the Resizing action in other file systems can destroy the data on the device you are working on.

Example 13.9. A playbook that resizes existing mylv1 and myvl2 logical volumes in the myvg volume group

```yaml
- hosts: all
  vars:
    storage_pools:
      - name: myvg
disks:
        - /dev/sda
        - /dev/sdb
        - /dev/sdc
volumes:
```
- name: mylv1
  size: 10 GiB
  fs_type: ext4
  mount_point: /opt/mount1
- name: mylv2
  size: 50 GiB
  fs_type: ext4
  mount_point: /opt/mount2

- name: Create LVM pool over three disks
  include_role:
    name: rhel-system-roles.storage

  - This playbook resizes the following existing file systems:
    - The Ext4 file system on the mylv1 volume, which is mounted at /opt/mount1, resizes to 10 GiB.
    - The Ext4 file system on the mylv2 volume, which is mounted at /opt/mount2, resizes to 50 GiB.

Additional resources

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

13.11. EXAMPLE ANSIBLE PLAYBOOK TO CREATE A SWAP PARTITION USING THE STORAGE RHEL SYSTEM ROLE

This section provides an example Ansible playbook. This playbook applies the storage role to create a swap partition, if it does not exist, or to modify the swap partition, if it already exists, on a block device using the default parameters.

Example 13.10. A playbook that creates or modify an existing XFS on /dev/sdb

---
- name: Create a disk device with swap
  - hosts: all
    vars:
      storage_volumes:
        - name: swap_fs
          type: disk
          disks:
            - /dev/sdb
          size: 15 GiB
          fs_type: swap
          roles:
            - rhel-system-roles.storage

  - The volume name (swap_fs in the example) is currently arbitrary. The storage role identifies the volume by the disk device listed under the disks: attribute.
13.12. CONFIGURING A RAID VOLUME USING THE STORAGE SYSTEM ROLE

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

Prerequisites

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

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

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

- You have an inventory file detailing the systems on which you want to deploy a RAID volume using the storage System Role.

Procedure

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

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

     roles:
       - name: rhel-system-roles.storage
   ```

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

3. Run the playbook on your inventory file:

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

Additional resources

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

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

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

**Prerequisites**

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

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

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

- You have an inventory file detailing the systems on which you want to configure an LVM pool with RAID using the `storage` System Role.

**Procedure**

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

NOTE

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


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

3. Run the playbook on your inventory file:

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

Additional resources

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

13.14. CREATING A LUKS ENCRYPTED VOLUME USING THE STORAGE ROLE

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

Prerequisites

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

NOTE

You do not have to have Red Hat Ansible Automation Platform installed on the systems on which you want to create the volume.
You have the rhel-system-roles package installed on the Ansible controller.

You have an inventory file detailing the systems on which you want to deploy a LUKS encrypted volume using the storage System Role.

Procedure

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

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

2. Optional: Verify playbook syntax:

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

3. Run the playbook on your inventory file:

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

Additional resources

- Encrypting block devices using LUKS
- `/usr/share/ansible/roles/rhel-system-roles.storage/README.md` file

13.15. ADDITIONAL RESOURCES

- For more information, install the rhel-system-roles package and see the following directories:
  - `/usr/share/doc/rhel-system-roles-VERSION/storage/`
  - `/usr/share/ansible/roles/rhel-system-roles.storage/`
CHAPTER 14. CONFIGURING TIME SYNCHRONIZATION USING RHEL SYSTEM ROLES

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

14.1. THE TIMESYNC SYSTEM ROLE

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

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

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

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

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

WARNING

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

Prerequisites

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

  NOTE

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

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

- You have an inventory file which lists the systems on which you want to deploy timesync System Role.
Procedure

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

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

2. Optional: Verify playbook syntax.

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

3. Run the playbook on your inventory file:

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

14.3. TIMESYNC SYSTEM ROLES VARIABLES

You can pass the following variable to the timesync role:

- `timesync_ntp_servers`:

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

Additional resources

- For a detailed reference on timesync role variables, install the rhel-system-roles package, and see the README.md files in the `/usr/share/doc/rhel-system-roles-VERSION/timesync` directory.
CHAPTER 15. MONITORING PERFORMANCE USING RHEL SYSTEM ROLES

As a system administrator, you can use the metrics RHEL System Role with any Ansible Automation Platform control node to monitor the performance of a system.

15.1. 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 15.1. Metrics system role variables

<table>
<thead>
<tr>
<th>Role variable</th>
<th>Description</th>
<th>Example usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>metrics_monitored_hosts</td>
<td>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 <code>/var/log</code> for each host.</td>
<td><code>metrics_monitored_hosts: [&quot;webserver.example.com&quot;, &quot;database.example.com&quot;]</code></td>
</tr>
<tr>
<td>metrics_retention_days</td>
<td>Configures the number of days for performance data retention before deletion.</td>
<td><code>metrics_retention_days: 14</code></td>
</tr>
<tr>
<td>metrics_graph_service</td>
<td>A boolean flag that enables the host to be set up with services for performance data visualization via <code>pcp</code> and <code>grafana</code>. Set to false by default.</td>
<td><code>metrics_graph_service: false</code></td>
</tr>
<tr>
<td>metrics_query_service</td>
<td>A boolean flag that enables the host to be set up with time series query services for querying recorded <code>pcp</code> metrics via <code>redis</code>. Set to false by default.</td>
<td><code>metrics_query_service: false</code></td>
</tr>
<tr>
<td>metrics_provider</td>
<td>Specifies which metrics collector to use to provide metrics. Currently, <code>pcp</code> is the only supported metrics provider.</td>
<td><code>metrics_provider: &quot;pcp&quot;</code></td>
</tr>
</tbody>
</table>

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

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

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

Procedure
1. Configure localhost in the /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.

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

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

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

Procedure

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

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

2. Create an Ansible playbook with the following content:

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

3. Run the Ansible playbook:

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

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

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

Prerequisites

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

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

Procedure

1. Create an Ansible playbook with the following content:

   ```
   ---
   - hosts: localhost
     vars:
       metrics_graph_service: yes
       metrics_query_service: yes
       metrics_retention_days: 10
   ```
2. Run the Ansible playbook:

```bash
# 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](#).

### 15.5. 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**

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

**Procedure**

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

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

2. Run the Ansible playbook:

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

**Verification steps**
15.6. 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**

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

   ```yaml
   ---
   - hosts: localhost
     roles:
       - role: rhel-system-roles.metrics
     vars:
       metrics_from_sql: 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
  ```

- 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
  ```
Additional resources

- **For more information about using Performance Co-Pilot for Microsoft SQL Server, see this Red Hat Developers Blog post.**
CHAPTER 16. 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.

16.1. 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

16.2. 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:

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

- 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.

16.3. DEPLOYING THE TLOG RHEL SYSTEM ROLE

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

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

Procedure

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

   ```yaml
   ---
   - 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, `notall` 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.

   ```bash
   # ansible-playbook --syntax-check playbook.yml
   
   # 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:
   ```sh
   # cd /etc/sssd/conf.d
   ```
2. Check the file content:
   ```sh
   # cat /etc/sssd/conf.d/sssd-session-recording.conf
   ```

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

### 16.4. 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.
- You have one control node, which is a system from which the Red Hat Ansible Engine configures the other systems.
- You have Red Hat Ansible Engine installed on the control node, from which you want to run the playbook.
- You have the `rhel-system-roles` package installed on the control node.
- You have at least one system on which you want to configure the `tlog` System Role. You do not have to have Red Hat Ansible Automation Platform installed on the systems on which you want to deploy the `tlog` solution.

**Procedure**

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

   ```yaml
   ---
   - name: Deploy session recording excluding users and groups
     hosts: all
     vars:
       tlog_scope_sssd: all
       tlog_exclude_users_sssd:
   ```
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;

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

3. Run the playbook on your inventory file:

   ```bash
   # 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:

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

2. Check the file content:

   ```bash
   # 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-VERSION/tlog/ and /usr/share/ansible/roles/rhel-system-roles.tlog/ directories.
- See Section 16.5, “Recording a session using the deployed tlog system role in the CLI”

16.5. 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

## 16.6. 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.