Red Hat Enterprise Linux 9

RHEL システムロールを使用したシステム管理の自動化

Red Hat Ansible Automation Platform Playbook を使用して複数のホストに RHEL をデプロイするための一貫性および反復性のある設定

Last Updated: 2024-04-16

Red Hat Enterprise Linux 9 RHEL システムロールを使用したシステム管理の自動化

Red Hat Ansible Automation Platform Playbook を使用して複数のホストに RHEL をデプロイするための一貫性および反復性のある設定

法律上の通知

概要

Red Hat Enterprise Linux (RHEL) システムロールは、一貫性および反復性のある RHEL システム管理を自動化するのに役立つ Ansible ロール、モジュール、および Playbook のコレクションです。RHEL システムロールを使用すると、単一のシステムから設定 Playbook を実行して、システムの大規模なインベントリーを効率的に管理できます。

目次

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

RED HAT ドキュメントへのフィードバック (英語のみ)

Red Hat ドキュメントに関するご意見やご感想をお寄せください。また、改善点があればお知らせください。

Jira からのフィードバック送信 (アカウントが必要)

1. Jira の Web サイトにログインします。
2. 上部のナビゲーションバーで **Create** をクリックします。
3. **Summary** フィールドにわかりやすいタイトルを入力します。
4. **Description** フィールドに、ドキュメントの改善に関するご意見を記入してください。ドキュメントの該当部分へのリンクも追加してください。
5. ダイアログの下部にある **Create** をクリックします。

第1章 RHEL システムロールの概要

RHEL システムロールは、Ansible ロールおよびモジュールのコレクションです。RHEL システムロールを使用すると、複数の RHEL メジャーバージョンにわたる複数の RHEL システムのシステム設定をリモートで管理できます。これを使用してシステムを設定するには、次のコンポーネントを使用する必要があります。

コントロールノード

コントロールノードは、Ansible コマンドと Playbook を実行するシステムです。コントロールノードには、Ansible Automation Platform、Red Hat Satellite、または RHEL 9、8、または 7 ホストを使用できます。詳細は、Preparing a control node on RHEL 9 を参照してください。

管理対象ノード

管理対象ノードは、Ansible で管理するサーバーとネットワークデバイスです。管理対象ノードは、ホストと呼ばれることもあります。管理対象ノードに Ansible をインストールする必要はありません。詳細は、管理対象ノードの準備 を参照してください。

Ansible Playbook

Playbook では、管理対象ノード上で実現したい設定、または管理対象ノード上のシステムが実行する一連の手順を定義します。Playbook は、Ansible の設定、デプロイメント、およびオーケストレーションの言語です。

Inventory

インベントリーファイルでは、管理対象ノードをリストし、各管理対象ノードの IP アドレスなどの情報を指定します。インベントリーでは、管理対象ノードを整理し、グループを作成およびネストして、スケーリングを容易にすることもできます。インベントリーファイルは、ホストファイルと呼ばれることもあります。

Red Hat Enterprise Linux9 では、**AppStream** リポジトリーで入手可能な **rhel-system-roles** パッケージによって提供される次のロールを使用できます。

ロール名	ロールの説明	章のタイトル
certificate	証明書の発行および更新	RHEL システムロールを使用した証明書の要求
cockpit	Web コンソール	Cockpit RHEL システムロールを使用した Web コンソールのインストールと設定
crypto_policies	システム全体の暗号化ポリシー	システム間でのカスタム暗号化ポリシーの設定
ファイアウォール (firewall)	Firewalld	システムロールを使用した firewalld の設定
ha_cluster	HA クラスター	システムロールを使用した高可用性クラスターの設定
kdump	カーネルダンプ	RHEL システムロールを使用した kdump の設定
kernel_settings	カーネル設定	Ansible ロールを使用したカーネルパラメーターの永続的な設定

ロール名	ロールの説明	章のタイトル
logging	ロギング	ロギングシステムロールの使用
metrics	メトリック (PCP)	RHEL システムロールを使用したパフォーマンスの監視
network	ネットワーク	network RHEL システムロールを使用した InfiniBand 接続の管理
nbde_client	ネットワークバインドディスク暗号化クライアント	nbde_client および nbde_server システムロールの使用
nbde_server	ネットワークバインドディスク暗号化サーバー	nbde_client および nbde_server システムロールの使用
postfix	postfix	システムロールの postfix ロールの変数
postgresql	PostgreSQL	postgresql RHEL システムロールを使用した PostgreSQL のインストールと設定
selinux	SELinux	システムロールを使用した SELinux の設定
ssh	SSH クライアント	ssh システムロールを使用したセキュアな通信の設定
sshd	SSH サーバー	ssh システムロールを使用したセキュアな通信の設定
storage	ストレージ	RHEL システムロールを使用したローカルストレージの管理
tlog	ターミナルセッションの記録	tlog RHEL システムロールを使用したセッション記録用システムの設定
timesync	時刻同期	RHEL システムロールを使用した時刻同期の設定
vpn	VPN	vpn RHEL システムロールを使用した IPsec による VPN 接続の設定

関連情報

- Red Hat Enterprise Linux (RHEL) system roles
- **/usr/share/ansible/roles/rhel-system-roles.<role_name>/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/<role_name>/** ディレクトリー

第2章 RHEL システムロールを使用するためのコントロールノードと管理対象ノードの準備

個々の RHEL システムロールを使用してサービスと設定を管理するには、その前に、コントロールノードと管理対象ノードを準備する必要があります。

2.1. RHEL 9 でのコントロールノードの準備

RHEL システムロールを使用する前に、コントロールノードを設定する必要があります。次に、このシステムは、Playbook に従ってインベントリーから管理対象ホストを設定します。

前提条件

- システムはカスタマーポータルに登録されます。
- **Red Hat Enterprise Linux Server** サブスクリプションがシステムにアタッチされている。
- オプション: **Ansible Automation Platform** サブスクリプションがシステムにアタッチされている。

手順

1. **rhel-system-roles** パッケージをインストールします。

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

   このコマンドは、**ansible-core** パッケージを依存関係としてインストールします。

2. Playbook を管理および実行するための **ansible** という名前のユーザーを作成します。

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

3. 新しく作成した **ansible** ユーザーに切り替えます。

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

   このユーザーとして残りの手順を実行します。

4. SSH の公開鍵と秘密鍵を作成します。

```
[ansible@control-node]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ansible/.ssh/id_rsa):
Enter passphrase (empty for no passphrase): <password>
Enter same passphrase again: <password>
...
```

   キーファイルの推奨されるデフォルトの場所を使用します。

5. オプション: 接続を確立するたびに Ansible が SSH キーのパスワードを要求しないように、SSH エージェントを設定します。

6. **~/.ansible.cfg** ファイルを次の内容で作成します。

```
[defaults]
inventory = /home/ansible/inventory
remote_user = ansible

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

注記

~/.ansible.cfg ファイルの設定は優先度が高く、グローバルな **/etc/ansible/ansible.cfg** ファイルの設定をオーバーライドします。

これらの設定を使用して、Ansible は次のアクションを実行します。

- 指定されたインベントリーファイルでホストを管理します。
- 管理対象ノードへの SSH 接続を確立するときに、**remote_user** パラメーターで設定されたアカウントを使用します。
- **sudo** ユーティリティーを使用して、**root** ユーザーとして管理対象ノードでタスクを実行します。
- Playbook を適用するたびに、リモートユーザーの root パスワードの入力を求められます。これは、セキュリティー上の理由から推奨されます。

7. 管理対象ホストのホスト名をリストする **~/inventory** ファイルを INI または YAML 形式で作成します。インベントリーファイルでホストのグループを定義することもできます。たとえば、以下は、3 つのホストと **US** という名前の 1 つのホストグループを含む INI 形式のインベントリーファイルです。

```
managed-node-01.example.com

[US]
managed-node-02.example.com ansible_host=192.0.2.100
managed-node-03.example.com
```

コントロールノードはホスト名を解決できる必要があることに注意してください。DNS サーバーが特定のホスト名を解決できない場合は、ホストエントリーの横に **ansible_host** パラメーターを追加して、その IP アドレスを指定します。

次のステップ

- 管理対象ノードを準備します。詳細は、管理対象ノードの準備 を参照してください。

関連情報

- RHEL 9 および RHEL 8.6 以降の AppStream リポジトリーに含まれる Ansible Core パッケージのサポート範囲
- subscription-manager を使用して Red Hat カスタマーポータルにシステムを登録してサブスクライブする

- **ssh-keygen(1)** man ページ
- ssh-agent を使用して SSH キーでリモートマシンに接続する手順
- Ansible 構成設定
- インベントリーの構築方法

2.2. 管理対象ノードの準備

管理対象ノードはインベントリーにリストされているシステムであり、Playbook に従ってコントロールノードによって設定されます。管理対象ホストに Ansible をインストールする必要はありません。

前提条件

- コントロールノードを準備している。詳細は、Preparing a control node on RHEL 9 を参照してください。
- コントロールノードから SSH アクセスできる。

重要

root ユーザーとしての直接 SSH アクセスはセキュリティーリスクを引き起こします。このリスクを軽減するには、管理対象ノードを準備するときに、このノード上にローカルユーザーを作成し、**sudo** ポリシーを設定します。続いて、コントロールノードの Ansible は、ローカルユーザーアカウントを使用して管理対象ノードにログインし、**root** などの別のユーザーとして Playbook を実行できます。

手順

1. **ansible** という名前のユーザーを作成します。

```
[root@managed-node-01]# useradd ansible
```

   コントロールノードは後でこのユーザーを使用して、このホストへの SSH 接続を確立します。

2. **ansible** ユーザーのパスワードを設定します。

```
[root@managed-node-01]# passwd ansible
Changing password for user ansible.
New password: <password>
Retype new password: <password>
passwd: all authentication tokens updated successfully.
```

   Ansible が **sudo** を使用して **root** ユーザーとしてタスクを実行する場合は、このパスワードを入力する必要があります。

3. **ansible** ユーザーの SSH 公開鍵を管理対象ノードにインストールします。

   a. **ansible** ユーザーとしてコントロールノードにログインし、SSH 公開鍵を管理対象ノードにコピーします。

```
[ansible@control-node]$ ssh-copy-id managed-node-01.example.com
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed:
```

```
"/home/ansible/.ssh/id_rsa.pub"
The authenticity of host 'managed-node-01.example.com (192.0.2.100)' can't be
established.
ECDSA key fingerprint is
SHA256:9bZ33GJNODK3zbNhybokN/6Mq7hu3vpBXDrCxe7NAvo.
```

b. プロンプトが表示されたら、**yes** と入力して接続します。

```
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that
are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is
to install the new keys
```

c. プロンプトが表示されたら、パスワードを入力します。

```
ansible@managed-node-01.example.com's password: <password>

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh '<managed-node-01.example.com>'"
and check to make sure that only the key(s) you wanted were added.
```

d. コントロールノードでコマンドをリモートで実行して、SSH 接続を確認します。

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

4. **ansible** ユーザーの **sudo** 設定を作成します。

a. **visudo** コマンドを使用して、**/etc/sudoers.d/ansible** ファイルを作成および編集します。

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

通常のエディターと比べて **visudo** を使用する利点は、このユーティリティーがファイルをインストールする前に基本的な健全性チェックと解析エラーのチェックを提供することです。

b. **/etc/sudoers.d/ansible** ファイルで、要件に応じた **sudoers** ポリシーを設定します。次に例を示します。

- **ansible** ユーザーのパスワードを入力した後、このホスト上で任意のユーザーおよびグループとしてすべてのコマンドを実行する権限を **ansible** ユーザーに付与するには、以下を使用します。

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

- **ansible** ユーザーのパスワードを入力せずに、このホスト上で任意のユーザーおよびグループとしてすべてのコマンドを実行する権限を **ansible** ユーザーに付与するには、以下を使用します。

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

または、セキュリティー要件に合わせてより細かいポリシーを設定します。**sudoers** ポリシーの詳細は、**sudoers(5)** man ページを参照してください。

検証

1. すべての管理対象ノード上のコントロールノードからコマンドを実行できることを確認します。

```
[ansible@control-node]$ ansible all -m ping
BECOME password: <password>
managed-node-01.example.com | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
...
```

ハードコーディングされたすべてのホストグループには、インベントリーファイルにリストされているすべてのホストが動的に含まれます。

2. Ansible **command** モジュールを使用して管理対象ホスト上で **whoami** ユーティリティーを実行し、権限昇格が正しく機能することを確認します。

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

コマンドが root を返した場合、管理対象ノード上で **sudo** が正しく設定されています。

関連情報

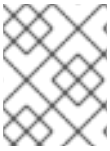
- RHEL 9 でのコントロールノードの準備
- **sudoers(5)** man ページ

第3章 コレクションのインストールおよび使用

Ansible コレクションは、新たな方法で自動化を配布、メンテナンス、および使用します。Playbook、ロール、モジュール、プラグインなど、複数のタイプの Ansible コンテンツを組み合わせることで、柔軟性とスケーラビリティーが向上します。

Ansible Collections は、従来の RHEL システムロール形式に対する選択肢の 1 つです。Ansible Collection 形式の RHEL システムロールを使用することは、従来の形式の RHEL システムロールを使用するのとほぼ同じです。相違点は、Ansible Collections は **fully qualified collection name** (FQCN) という概念を使用する点です。このコレクション名は、**namespace** と **collection name** で構成されます。使用する **namespace** は **redhat** で、**collection name** は **rhel_system_roles** です。そのため、**kernel_settings** ロールの従来の RHEL システムロール形式は、**rhel-system-roles.kernel_settings** (ダッシュ付き) で表されますが、**kernel_settings** ロールに Collection の **fully qualified collection name** を使用した場合は、**redhat.rhel_system_roles.kernel_settings** (アンダースコア付き) で表されます。

namespace と **collection name** を組み合わせると、確実にオブジェクトが一意になります。また、オブジェクトが競合せずに Ansible Collections および namespace 間で共有されます。



注記

Automation Hub にアクセスして Red Hat 認定コレクションを使用するには、Ansible Automation Platform (AAP サブスクリプション) が必要です。

以下は、FQCN 形式の **logging** RHEL システムロールを適用するサンプル Playbook です。

```
---
- name: Deploying basics input and implicit files output
  hosts: managed-node-01.example.com
  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]
```

3.1. コレクションの構造

コレクションは、Ansible コンテンツのパッケージ形式です。データ構造は以下のようになります。

- docs/: 例も含めてコレクションについてまとめたローカルドキュメント。(ロールがドキュメントを提供する場合)
- galaxy.yml: Ansible Collection パッケージに含まれる MANIFEST.json のソースデータ
- Playbook/: Playbook はこちらで利用できます。

    - tasks/: include_tasks/import_tasks の使用状況に関する task list files を保管します。
- plugins/: Ansible プラグインおよびモジュールはすべてこちらの各サブディレクトリーから入手できます。
    - modules/: Ansible モジュール
    - modules_utils/: モジュール開発用の共通コード
    - lookup/: プラグインの検索
    - filter/: Jinja2 フィルタープラグイン
    - connection/: 接続プラグインはデフォルトを使用していない場合に必要です。
- roles/: Ansible ロール用ディレクトリー
- tests/: コレクションの内容のテスト

3.2. CLI を使用したコレクションのインストール

コレクションは、Playbook、ロール、モジュール、およびプラグインなど、Ansible コンテンツのディストリビューション形式です。

コレクションは、Ansible Galaxy、ブラウザーまたはコマンドラインを使用してインストールできます。

前提条件

- コントロールノードと管理対象ノードを準備している。

手順

- RPM パッケージからコレクションをインストールします。

```
# dnf install rhel-system-roles
```

インストールが完了すると、ロールを **redhat.rhel_system_roles.<role_name>** 形式で利用できるようになります。

検証

- インストールを確認します。

```
$ ansible-playbook -c local -i localhost, --check --become
/usr/share/ansible/collections/ansible_collections/redhat/rhel_system_roles/tests/kernel_settings
ests_default.yml
```

コマンド出力の最後の行には、値 **failed=0** が含まれている必要があります。

ローカルホストで **check** モードで **kernel_settings** ロールを実行する必要があります。ただし、**kernel_settings** ロールは **--check** モードでは機能しません。これを機能させるには、Playbook 内の **service** タスクと **config** タスクを、**--check** モードのときにスキップするように変更してください。Ansible **package** モジュールに必要な **--become** パラメーターも使用する必要があります。ただし、このパラメーターはシステムを変更しません。

注記

localhost の後のコンマは必須です。リストにホストが 1 つしかない場合でも、追加する必要があります。これがないと、**ansible-playbook** は **localhost** をファイルまたはディレクトリーとして識別します。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.<role_name>/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/<role_name>/** ディレクトリー

3.3. AUTOMATION HUB からのコレクションのインストール

Automation Hub を使用している場合は、Automation Hub でホストされている RHEL システムロールコレクションをインストールできます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- Ansible Automation Platform (AAP サブスクリプション) を持っている。

手順

1. **ansible.cfg** 設定ファイルでコンテンツのデフォルトソースとして Red Hat Automation Hub を定義します。コンテンツについては、プライマリーソースとしての Red Hat Automation Hub の設定 を参照してください。
2. Automation Hub から **redhat.rhel_system_roles** コレクションをインストールします。

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

インストールが完了すると、ロールを **redhat.rhel_system_roles.<role_name>** 形式で利用できるようになります。

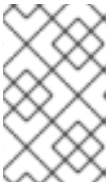
検証

- インストールを確認します。

```
$ ansible-playbook -c local -i localhost, --check --become
/usr/share/ansible/collections/ansible_collections/redhat/rhel_system_roles/tests/kernel_settings
ests_default.yml
```

コマンド出力の最後の行には、値 **failed=0** が含まれている必要があります。

ローカルホストで **check** モードで **kernel_settings** ロールを実行する必要があります。Ansible **package** モジュールに必要な **--become** パラメーターも使用する必要があります。ただし、このパラメーターはシステムを変更しません。



注記

localhost の後のコンマは必須です。リストにホストが 1 つしかない場合でも、追加する必要があります。これがないと、**ansible-playbook** は **localhost** をファイルまたはディレクトリーとして識別します。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.<role_name>/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/<role_name>/** ディレクトリー

第4章 RHEL の ANSIBLE IPMI モジュール

4.1. RHEL_MGMT コレクション

Intelligent Platform Management Interface (IPMI) は、ベースボード管理コントローラー (BMC) デバイスと通信するための一連の標準プロトコルの仕様です。**IPMI** モジュールを使用すると、ハードウェア管理の自動化を有効にしてサポートできます。**IPMI** モジュールは次の場所で使用できます。

- **rhel_mgmt** コレクション。パッケージ名は **ansible-collection-redhat-rhel_mgmt** です。
- 新しい **ansible-collection-redhat-rhel_mgmt** パッケージの一部である RHEL 8 AppStream。

次の IPMI モジュールが rhel_mgmt コレクションで使用可能です。

- **ipmi_boot**: ブートデバイスの順序の管理
- **ipmi_power**: マシンの電力管理

IPMI モジュールに使用される必須パラメーターは次のとおりです。

- **ipmi_boot** パラメーター:

モジュール名	説明
name	BMC のホスト名または IP アドレス。
password	BMC に接続するためのパスワード
bootdev	次回起動時に使用するデバイス * network * floppy * hd * safe * optical * setup * default
ユーザー	BMC に接続するためのユーザー名

- **ipmi_power** パラメーター:

モジュール名	説明
name	BMC ホスト名または IP アドレス

モジュール名	説明
password	BMC に接続するためのパスワード
user	BMC に接続するためのユーザー名
State	マシンが目的のステータスにあるかどうかを確認します * on * off * shutdown * reset * boot

4.2. IPMI_BOOT モジュールの使用

次の例は、Playbook で **ipmi_boot** モジュールを使用して、次回の起動用に起動デバイスを設定する方法を示しています。わかりやすくするために、ここに示す例では Ansible コントロールホストおよびマネージドホストと同じホストを使用しているため、Playbook が実行されるのと同じホストでモジュールを実行します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- **ansible-collection-redhat-rhel_mgmt** パッケージがインストールされている。
- **python3-pyghmi** パッケージが、コントロールノードまたは管理対象ノードのいずれかにインストールされている。
- 制御する IPMI BMC に、コントロールノードまたは管理対象ホスト (管理対象ホストとして **localhost** を使用していない場合) からネットワーク経由でアクセスできる。モジュールが IPMI プロトコルを使用してネットワーク経由で BMC に接続するため、通常、モジュールによって BMC が設定されているホストは、管理対象ホストとは異なることに注意してください。
- 適切なレベルのアクセスで BMC にアクセスするためのクレデンシャルがあります。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Set boot device to be used on next boot
  hosts: managed-node-01.example.com
```

```
tasks:
  - name: Ensure boot device is HD
    redhat.rhel_mgmt.ipmi_boot:
      user: <admin_user>
      password: <password>
      bootdev: hd
```

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

- Playbook を実行すると、Ansible が **success** を返します。

関連情報

- **/usr/share/ansible/collections/ansible_collections/redhat/rhel_mgmt/README.md** ファイル

4.3. IPMI_POWER モジュールの使用

この例は、Playbook で **ipmi_boot** モジュールを使用して、システムがオンになっているかどうかを確認する方法を示しています。わかりやすくするために、ここに示す例では Ansible コントロールホストおよびマネージドホストと同じホストを使用しているため、Playbook が実行されるのと同じホストでモジュールを実行します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- **ansible-collection-redhat-rhel_mgmt** パッケージがインストールされている。
- **python3-pyghmi** パッケージが、コントロールノードまたは管理対象ノードのいずれかにインストールされている。
- 制御する IPMI BMC に、コントロールノードまたは管理対象ホスト (管理対象ホストとして **localhost** を使用していない場合) からネットワーク経由でアクセスできる。モジュールが IPMI プロトコルを使用してネットワーク経由で BMC に接続するため、通常、モジュールによって BMC が設定されているホストは、管理対象ホストとは異なることに注意してください。
- 適切なレベルのアクセスで BMC にアクセスするためのクレデンシャルがあります。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Power management
  hosts: managed-node-01.example.com
  tasks:
    - name: Ensure machine is powered on
      redhat.rhel_mgmt.ipmi_power:
        user: <admin_user>
        password: <password>
        state: on
```

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

- Playbook を実行すると、Ansible が **true** を返します。

関連情報

- **/usr/share/ansible/collections/ansible_collections/redhat/rhel_mgmt/README.md** ファイル

第5章 RHEL の REDFISH モジュール

デバイスのリモート管理用の Redfish モジュールは、**redhat.rhel_mgmt** Ansible コレクションの一部になりました。Redfish モジュールを使用すると、標準の HTTPS トランスポートと JSON 形式を使用して、サーバーに関する情報を取得したり、帯域外 (OOB) コントローラーを介してそれらを制御したりして、ベアメタルサーバーとプラットフォームハードウェアで管理の自動化を簡単に使用できます。

5.1. REDFISH モジュール

redhat.rhel_mgmt Ansible コレクションは、Redfish 上の Ansible でのハードウェア管理をサポートする Redfish モジュールを提供します。**redhat.rhel_mgmt** コレクションは **ansible-collection-redhat-rhel_mgmt** パッケージで利用できます。インストールするには、CLI を使用した redhat.rhel_mgmt コレクションのインストール を参照してください。

次の Redfish モジュールは、**redhat.rhel_mgmt** コレクションで利用できます。

1. **redfish_info**: **redfish_info** モジュールは、システムインベントリーなどのリモートアウトオブバンド (OOB) コントローラーに関する情報を取得します。
2. **redfish_command**: **redfish_command** モジュールは、ログ管理やユーザー管理などの帯域外 (OOB) コントローラー操作と、システムの再起動、電源のオンとオフなどの電源操作を実行します。
3. **redfish_config**: **redfish_config** モジュールは、OOB 設定の変更や BIOS 設定の設定などの OOB コントローラー操作を実行します。

5.2. REDFISH モジュールのパラメーター

Redfish モジュールに使用されるパラメーターは次のとおりです。

redfish_info パラメーター:	説明
baseuri	(必須) - OOB コントローラーのベース URI。
category	(必須) - OOB コントローラーで実行するカテゴリーのリスト。デフォルト値は ["Systems"] です。
command	(必須) - OOB コントローラーで実行するコマンドのリスト。
username	OOB コントローラーへの認証用のユーザー名。
password	OOB コントローラーへの認証用のパスワード。

redfish_command パラメーター:	説明
baseuri	(必須) - OOB コントローラーのベース URI。
category	(必須) - OOB コントローラーで実行するカテゴリーのリスト。デフォルト値は ["Systems"] です。

redfish_command パラメーター:	説明
command	(必須) - OOB コントローラーで実行するコマンドのリスト。
username	OOB コントローラーへの認証用のユーザー名。
password	OOB コントローラーへの認証用のパスワード。

redfish_config パラメーター:	説明
baseuri	(必須) - OOB コントローラーのベース URI。
category	(必須) - OOB コントローラーで実行するカテゴリーのリスト。デフォルト値は ["Systems"] です。
command	(必須) - OOB コントローラーで実行するコマンドのリスト。
username	OOB コントローラーへの認証用のユーザー名。
password	OOB コントローラーへの認証用のパスワード。
bios_attributes	更新する BIOS 属性。

5.3. REDFISH_INFO モジュールの使用

次の例は、Playbook で **redfish_info** モジュールを使用して CPU インベントリーに関する情報を取得する方法を示しています。わかりやすくするために、ここに示す例では Ansible コントロールホストおよび管理対象ホストと同じホストを使用しているため、Playbook が実行されるのと同じホストでモジュールを実行します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- **ansible-collection-redhat-rhel_mgmt** パッケージがインストールされている。
- **python3-pyghmi** パッケージが、コントロールノードまたは管理対象ノードのいずれかにインストールされている。

- OOB コントローラーアクセスの詳細。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Manage out-of-band controllers using Redfish APIs
  hosts: managed-node-01.example.com
  tasks:
    - name: Get CPU inventory
      redhat.rhel_mgmt.redfish_info:
        baseuri: "<URI>"
        username: "<username>"
        password: "<password>"
        category: Systems
        command: GetCpuInventory
      register: result
```

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

- Playbook を実行すると、Ansible が CPU インベントリーの詳細を返します。

関連情報

- **/usr/share/ansible/collections/ansible_collections/redhat/rhel_mgmt/README.md** ファイル

5.4. REDFISH_COMMAND モジュールの使用

次の例は、playbook で **redfish_command** モジュールを使用してシステムをオンにする方法を示しています。わかりやすくするために、ここに示す例では Ansible コントロールホストおよび管理対象ホストと同じホストを使用しているため、Playbook が実行されるのと同じホストでモジュールを実行します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

- **ansible-collection-redhat-rhel_mgmt** パッケージがインストールされている。
- **python3-pyghmi** パッケージが、コントロールノードまたは管理対象ノードのいずれかにインストールされている。
- OOB コントローラーアクセスの詳細。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- name: Manage out-of-band controllers using Redfish APIs
  hosts: managed-node-01.example.com
  tasks:
    - name: Power on system
      redhat.rhel_mgmt.redfish_command:
        baseuri: "<URI>"
        username: "<username>"
        password: "<password>"
        category: Systems
        command: PowerOn
```

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

- システムの電源がオンになります。

関連情報

- **/usr/share/ansible/collections/ansible_collections/redhat/rhel_mgmt/README.md** ファイル

5.5. REDFISH_CONFIG モジュールの使用

次の例は、Playbook で **redfish_config** モジュールを使用して、UEFI で起動するようにシステムを設定する方法を示しています。わかりやすくするために、ここに示す例では Ansible コントロールホストおよび管理対象ホストと同じホストを使用しているため、Playbook が実行されるのと同じホストでモジュールを実行します。

前提条件

- コントロールノードと管理対象ノードを準備している。

- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- **ansible-collection-redhat-rhel_mgmt** パッケージがインストールされている。
- **python3-pyghmi** パッケージが、コントロールノードまたは管理対象ノードのいずれかにインストールされている。
- OOB コントローラーアクセスの詳細。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Manages out-of-band controllers using Redfish APIs
  hosts: managed-node-01.example.com
  tasks:
    - name: Set BootMode to UEFI
      redhat.rhel_mgmt.redfish_config:
        baseuri: "<URI>"
        username: "<username>"
        password: "<password>"
 category: Systems
        command: SetBiosAttributes
        bios_attributes:
          BootMode: Uefi
```

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

- システムの起動モードが UEFI に設定されます。

関連情報

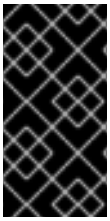
- **/usr/share/ansible/collections/ansible_collections/redhat/rhel_mgmt/README.md** ファイル

第6章 KERNEL_SETTINGS RHEL システムロールを使用したカーネルパラメーターの永続的な設定

kernel_settings RHEL システムロールを使用すると、複数のクライアントにカーネルパラメーターを一度に設定できます。この解決策は以下のとおりです。

- 効率的な入力設定を持つ使いやすいインターフェイスを提供します。
- すべてのカーネルパラメーターを 1 か所で保持します。

コントロールマシンから **kernel_settings** ロールを実行すると、カーネルパラメーターはすぐに管理システムに適用され、再起動後も維持されます。



重要

RHEL チャネルで提供される RHEL システムロールは、デフォルトの App Stream リポジトリー内の RPM パッケージとして RHEL のお客様に提供されることに注意してください。また、RHEL システムロールは、Ansible Automation Hub を介して Ansible サブスクリプションをご利用のお客様に、コレクションとして提供されます。

6.1. KERNEL_SETTINGS ロールの概要

RHEL システムロールは、複数のシステムをリモートで管理する、一貫した設定インターフェイスを提供する一連のロールです。

RHEL システムロールは、**kernel_settings** RHEL システムロールを使用してカーネルを自動的に設定するために導入されました。**rhel-system-roles** パッケージには、このシステムロールと参考ドキュメントも含まれます。

カーネルパラメーターを自動的に 1 つ以上のシステムに適用するには、Playbook で選択したロール変数を 1 つ以上使用して、**kernel_settings** ロールを使用します。Playbook は人間が判読でき、YAML 形式で記述される 1 つ以上のプレイのリストです。

インベントリーファイルを使用して、Ansible が Playbook に従って設定するシステムセットを定義することができます。

kernel_settings ロールを使用して、以下を設定できます。

- **kernel_settings_sysctl** ロールを使用したカーネルパラメーター
- **kernel_settings_sysfs** ロールを使用したさまざまなカーネルサブシステム、ハードウェアデバイス、およびデバイスドライバー
- **systemd** サービスマネージャーの CPU アフィニティー を、**kernel_settings_systemd_cpu_affinity** ロール変数を使用してフォーク処理します。
- **kernel_settings_transparent_hugepages** および **kernel_settings_transparent_hugepages_defrag** のロール変数を使用したカーネルメモリーサブシステムの Transparent Huge Page

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.kernel_settings/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/kernel_settings/** ディレクトリー

- Playbook の使用
- インベントリーの構築方法

6.2. KERNEL_SETTINGS ロールを使用した選択したカーネルパラメーターの適用

以下の手順に従って、Ansible Playbook を準備および適用し、複数の管理システムで永続化の影響でカーネルパラメーターをリモートに設定します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Configure kernel settings
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.kernel_settings
  vars:
    kernel_settings_sysctl:
      - name: fs.file-max
        value: 400000
      - name: kernel.threads-max
        value: 65536
    kernel_settings_sysfs:
      - name: /sys/class/net/lo/mtu
        value: 65000
    kernel_settings_transparent_hugepages: madvise
```

   - **name**: 任意の文字列をラベルとしてプレイに関連付け、プレイの対象を特定するオプションのキー。
   - **hosts**: プレイを実行するホストを指定するプレイ内のキー。このキーの値または値は、マネージドホストの個別名または **inventory** ファイルで定義されているホストのグループとして指定できます。
   - **vars**: 選択したカーネルパラメーター名と、それらに対して設定する必要がある値を含む変数のリストを表す Playbook のセクション。
   - **role**: **vars** セクションで指定されたパラメーターと値を設定する RHEL システムロールを指定するキー。

注記

必要に応じて、Playbook のカーネルパラメーターとその値を変更することができます。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

4. マネージドホストを再起動して、影響を受けるカーネルパラメーターをチェックし、変更が適用され、再起動後も維持されていることを確認します。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.kernel_settings/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/kernel_settings/** ディレクトリー
- Playbook の使用
- 変数の使用
- ロール

第7章 RHC システムロールを使用したシステムの登録

rhc RHEL システムロールを使用すると、管理者は Red Hat Subscription Management (RHSM) および Satellite サーバーへの複数のシステムの登録を自動化できます。このロールは、Ansible を使用することで、Insights 関連の設定タスクおよび管理タスクもサポートします。

7.1. RHC システムのロールの概要

RHEL システムロールは、複数のシステムをリモートで管理するための一貫した設定インターフェイスを提供するロールのセットです。リモートホスト設定 (**rhc**) RHEL システムロールを使用すると、管理者は RHEL システムを Red Hat Subscription Management (RHSM) および Satellite サーバーに簡単に登録できます。デフォルトでは、**rhc** RHEL システムロールを使用してシステムを登録すると、システムが Insights に接続されます。さらに、**rhc** RHEL システムロールを使用すると、次のことが可能になります。

- Red Hat Insights への接続の設定
- リポジトリーの有効化および無効化
- 接続に使用するプロキシーの設定
- Insights 修復と自動更新の設定
- システムのリリースの設定
- Insights タグの設定

7.2. RHC システムロールを使用したシステムの登録

rhc RHEL システムロールを使用して、システムを Red Hat に登録できます。デフォルトでは、 **rhc** RHEL システムロールは、登録時にシステムを Red Hat Insights に接続します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 機密性の高い変数を暗号化されたファイルに保存します。
   a. vault を作成します。

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

   b. **ansible-vault create** コマンドでエディターが開いたら、機密データを **<key>: <value>** 形式で入力します。

```
activationKey: <activation_key>
username: <username>
password: <password>
```

c. 変更を保存して、エディターを閉じます。Ansible は vault 内のデータを暗号化します。

2. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

- アクティベーションキーと組織 ID を使用して登録するには (推奨)、次の Playbook を使用します。

```
---
- name: Registering system using activation key and organization ID
  hosts: managed-node-01.example.com
  vars_files:
    - vault.yml
  roles:
    - role: rhel-system-roles.rhc
  vars:
    rhc_auth:
      activation_keys:
        keys:
          -  "{{ activationKey }}"
    rhc_organization: organizationID
```

- ユーザー名とパスワードを使用して登録するには、次の Playbook を使用します。

```
---
- name: Registering system with username and password
  hosts: managed-node-01.example.com
  vars_files:
    - vault.yml
  vars:
    rhc_auth:
      login:
        username: "{{ username }}"
        password: "{{ password }}"
  roles:
    - role: rhel-system-roles.rhc
```

3. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

4. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.rhc/README.md** ファイル

- **/usr/share/doc/rhel-system-roles/rhc/** ディレクトリー

7.3. RHC システムロールを使用した SATELLITE へのシステムの登録

組織が Satellite を使用してシステムを管理する場合、Satellite を介してシステムを登録する必要があります。**rhc** RHEL システムロールを使用して、システムを Satellite にリモートで登録できます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 機密性の高い変数を暗号化されたファイルに保存します。
   a. vault を作成します。

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

   b. **ansible-vault create** コマンドでエディターが開いたら、機密データを **<key>: <value>** 形式で入力します。

   ```
   activationKey: <activation_key>
   ```

   c. 変更を保存して、エディターを閉じます。Ansible は vault 内のデータを暗号化します。
2. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Register to the custom registration server and CDN
  hosts: managed-node-01.example.com
  vars_files:
    - vault.yml
  roles:
    - role: rhel-system-roles.rhc
  vars:
    rhc_auth:
      login:
        activation_keys:
          keys:
            - "{{ activationKey }}"
        rhc_organization: organizationID
    rhc_server:
      hostname: example.com
        port: 443
        prefix: /rhsm
    rhc_baseurl: http://example.com/pulp/content
```

3. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

4. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.rhc/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/rhc/** ディレクトリー

7.4. RHC システムロールを使用して登録後に INSIGHTS への接続を無効にする

rhc RHEL システムロールを使用してシステムを登録すると、このロールによりデフォルトで Red Hat Insights への接続が有効になります。必要ない場合は、**rhc** RHEL システムロールを使用して無効にできます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- システムを登録している。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- name: Disable Insights connection
  hosts: managed-node-01.example.com
  roles:
    - role: rhel-system-roles.rhc
  vars:
    rhc_insights:
      state: absent
```

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.rhc/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/rhc/** ディレクトリー

7.5. RHC システムロールを使用したリポジトリーの有効化

rhc RHEL システムロールを使用して、管理対象ノード上のリポジトリーをリモートで有効または無効にできます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- 管理対象ノード上で有効または無効にするリポジトリーの詳細を把握している。
- システムを登録している。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。
   - リポジトリーを有効にするには、以下を行います。

```
---
- name: Enable repository
  hosts: managed-node-01.example.com
  roles:
    - role: rhel-system-roles.rhc
  vars:
    rhc_repositories:
      - {name: "RepositoryName", state: enabled}
```

   - リポジトリーを無効にするには、以下を行います。

```
---
- name: Disable repository
  hosts: managed-node-01.example.com
  vars:
    rhc_repositories:
```

```
    - {name: "RepositoryName", state: disabled}
roles:
  - role: rhel-system-roles.rhc
```

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.rhc/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/rhc/** ディレクトリー

7.6. RHC システムロールを使用したリリースバージョンの設定

最新バージョンではなく、特定のマイナー RHEL バージョンのリポジトリーのみを使用するようにシステムを制限できます。このようにして、システムを特定のマイナー RHEL バージョンにロックできます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- システムをロックする RHEL のマイナーバージョンを把握している。システムをロックできるのは、ホストが現在実行している RHEL マイナーバージョン、またはそれ以降のマイナーバージョンのみである点に注意してください。
- システムを登録している。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- name: Set Release
  hosts: managed-node-01.example.com
  roles:
    - role: rhel-system-roles.rhc
  vars:
    rhc_release: "8.6"
```

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.rhc/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/rhc/** ディレクトリー

7.7. RHC システムロールを使用してホストを登録する際のプロキシーサーバーの使用

セキュリティー制限により、プロキシーサーバー経由でのみインターネットへのアクセスが許可されている場合は、**rhc** RHEL システムロールを使用してシステムを登録するときに、Playbook でプロキシーの設定を指定できます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 機密性の高い変数を暗号化されたファイルに保存します。
   a. vault を作成します。

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

   b. **ansible-vault create** コマンドでエディターが開いたら、機密データを **<key>: <value>** 形式で入力します。

```
username: <username>
password: <password>
proxy_username: <proxyusernme>
proxy_password: <proxypassword>
```

   c. 変更を保存して、エディターを閉じます。Ansible は vault 内のデータを暗号化します。

2. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

   - プロキシーを使用して RHEL カスタマーポータルに登録するには、以下を実行します。

```
---
- name: Register using proxy
  hosts: managed-node-01.example.com
  vars_files:
    - vault.yml
  roles:
    - role: rhel-system-roles.rhc
  vars:
    rhc_auth:
      login:
        username: "{{ username }}"
        password: "{{ password }}"
    rhc_proxy:
      hostname: proxy.example.com
      port: 3128
      username: "{{ proxy_username }}"
      password: "{{ proxy_password }}"
```

   - Red Hat Subscription Manager サービスの設定からプロキシーサーバーを削除するには、以下を実行します。

```
---
- name: To stop using proxy server for registration
  hosts: managed-node-01.example.com
  vars_files:
    - vault.yml
  vars:
    rhc_auth:
      login:
        username: "{{ username }}"
        password: "{{ password }}"
    rhc_proxy: {"state":"absent"}
  roles:
    - role: rhel-system-roles.rhc
```

3. Playbook の構文を検証します。

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

   このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

4. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.rhc/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/rhc/** ディレクトリー

7.8. RHC システムロールを使用した INSIGHTS ルールの自動更新の無効化

rhc RHEL システムロールを使用して、Red Hat Insights の自動収集ルール更新を無効にできます。デフォルトでは、システムを Red Hat Insights に接続すると、このオプションが有効になります。**rhc** RHEL システムロールを使用すると、これを無効にできます。

注記

この機能を無効にする場合は、古いルール定義ファイルを使用し、最新の検証更新を取得しないリスクがあります。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- システムを登録している。

手順

1. 機密性の高い変数を暗号化されたファイルに保存します。
   a. vault を作成します。

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

   b. **ansible-vault create** コマンドでエディターが開いたら、機密データを **<key>: <value>** 形式で入力します。

   ```
   username: <username>
   password: <password>
   ```

   c. 変更を保存して、エディターを閉じます。Ansible は vault 内のデータを暗号化します。
2. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Disable Red Hat Insights autoupdates
  hosts: managed-node-01.example.com
  vars_files:
    - vault.yml
  roles:
    - role: rhel-system-roles.rhc
  vars:
    rhc_auth:
      login:
        username: "{{ username }}"
        password: "{{ password }}"
```

```
    rhc_insights:
      autoupdate: false
      state: present
```

3. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

4. Playbook を実行します。

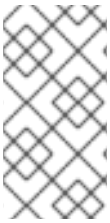
```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.rhc/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/rhc/** ディレクトリー

7.9. RHC RHEL システムロールを使用した INSIGHTS 修復の無効化

rhc RHEL システムロールを使用して、動的設定を自動的に更新するようにシステムを設定できます。システムを Red Hat Insights に接続すると、デフォルトで有効になります。必要ない場合は無効にすることができます。



注記

rhc RHEL システムロールを使用して修復を有効にすると、Red Hat に直接接続したときにシステムを修復する準備が完了します。Satellite または Capsule に接続されているシステムの場合は、修復を有効にするために別の方法を実行する必要があります。Red Hat Insights 修復の詳細は、Red Hat Insights 修復ガイド を参照してください。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- Insights 修復が有効になっている。
- システムを登録している。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- name: Disable remediation
  hosts: managed-node-01.example.com
```

```
roles:
  - role: rhel-system-roles.rhc
vars:
  rhc_insights:
    remediation: absent
    state: present
```

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.rhc/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/rhc/** ディレクトリー

7.10. RHC システムロールを使用した INSIGHTS タグの設定

タグを使用して、システムのフィルタリングとグループ化を行うことができます。要件に基づいて、タグをカスタマイズすることもできます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 機密性の高い変数を暗号化されたファイルに保存します。
   a. vault を作成します。

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

   b. **ansible-vault create** コマンドでエディターが開いたら、機密データを **<key>: <value>** 形式で入力します。

```
username: <username>
password: <password>
```

c. 変更を保存して、エディターを閉じます。Ansible は vault 内のデータを暗号化します。

2. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Creating tags
  hosts: managed-node-01.example.com
  vars_files:
    - vault.yml
  roles:
    - role: rhel-system-roles.rhc
  vars:
    rhc_auth:
      login:
        username: "{{ username }}"
        password: "{{ password }}"
    rhc_insights:
      tags:
        group: group-name-value
        location: location-name-value
        description:
          - RHEL8
          - SAP
        sample_key:value
      state: present
```

3. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

4. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.rhc/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/rhc/** ディレクトリー
- Red Hat Insights のシステムのフィルタリングとグループ

7.11. RHC システムロールを使用したシステムの登録解除

サブスクリプションサービスが不要になった場合は、Red Hat からシステムの登録を解除できます。

前提条件

- コントロールノードと管理対象ノードを準備している。

- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- システムはすでに登録されています。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Unregister the system
  hosts: managed-node-01.example.com
  roles:
    - role: rhel-system-roles.rhc
  vars:
    rhc_state: absent
```

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.rhc/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/rhc/** ディレクトリー

第8章 RHEL システムロールを使用したネットワーク設定

管理者は、**network** RHEL システムロールを使用して、ネットワーク関連の設定および管理タスクを自動化できます。

8.1. NETWORK RHEL システムロールとインターフェイス名を使用した静的 IP アドレスでのイーサネット接続設定

network RHEL システムロールを使用して、イーサネット接続をリモートで設定できます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- サーバーに、物理または仮想のイーサネットデバイスが設定されている。
- 管理対象ノードが NetworkManager を使用してネットワークを設定している。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure an Ethernet connection with static IP
      ansible.builtin.include_role:
        name: rhel-system-roles.network
      vars:
        network_connections:
          - name: enp1s0
            interface_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
            state: up
```

これらの設定では、次の設定を使用して **enp1s0** デバイスのイーサネット接続プロファイルを定義します。

- 静的 IPv4 アドレス: サブネットマスクが **/24** の **192.0.2.1**
- 静的 IPv6 アドレス - **2001:db8:1::1** (**/64** サブネットマスクあり)
- IPv4 デフォルトゲートウェイ - **192.0.2.254**
- IPv6 デフォルトゲートウェイ - **2001:db8:1::fffe**
- IPv4 DNS サーバー - **192.0.2.200**
- IPv6 DNS サーバー - **2001:db8:1::ffbb**
- DNS 検索ドメイン - **example.com**

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

8.2. NETWORK RHEL システムロールとデバイスパスを使用した静的 IP アドレスでのイーサネット接続設定

network RHEL システムロールを使用して、イーサネット接続をリモートで設定できます。

デバイスパスは、次のコマンドで識別できます。

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

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- サーバーに、物理または仮想のイーサネットデバイスが設定されている。
- 管理対象ノードが NetworkManager を使用してネットワークを設定している。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure an Ethernet connection with static IP
      ansible.builtin.include_role:
        name: rhel-system-roles.network
      vars:
        network_connections:
          - name: example
            match:
              path:
                - pci-0000:00:0[1-3].0
                - &!pci-0000:00:02.0
            type: ethernet
            autoconnect: yes
            ip:
              address:
                - 192.0.2.1/24
                - 2001:db8:1::1/64
              gateway4: 192.0.2.254
              gateway6: 2001:db8:1::fffe
              dns:
                - 192.0.2.200
                - 2001:db8:1::ffbb
              dns_search:
                - example.com
            state: up
```

これらの設定では、次の設定を使用してイーサネット接続プロファイルを定義します。

- 静的 IPv4 アドレス: サブネットマスクが /**24** の **192.0.2.1**
- 静的 IPv6 アドレス - **2001:db8:1::1** (/**64** サブネットマスクあり)
- IPv4 デフォルトゲートウェイ - **192.0.2.254**
- IPv6 デフォルトゲートウェイ - **2001:db8:1::fffe**
- IPv4 DNS サーバー - **192.0.2.200**
- IPv6 DNS サーバー - **2001:db8:1::ffbb**
- DNS 検索ドメイン - **example.com**
  この例の **match** パラメーターは、PCI ID **0000:00:0[1-3].0** に一致するデバイスには Ansible によってプレイを適用し、**0000:00:02.0** には適用しないことを定義します。使用できる特殊な修飾子およびワイルドカードの詳細は、**/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイルーの **match** パラメーターの説明を参照してください。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

8.3. NETWORK RHEL システムロールとインターフェイス名を使用した動的 IP アドレスでのイーサネット接続設定

network RHEL システムロールを使用して、イーサネット接続をリモートで設定できます。動的 IP アドレス設定との接続の場合、NetworkManager は、DHCP サーバーから接続の IP 設定を要求します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- サーバーに、物理または仮想のイーサネットデバイスが設定されている。
- DHCP サーバーをネットワークで使用できる。
- 管理対象ノードが NetworkManager を使用してネットワークを設定している。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure an Ethernet connection with dynamic IP
      ansible.builtin.include_role:
        name: rhel-system-roles.network
      vars:
        network_connections:
          - name: enp1s0
            interface_name: enp1s0
            type: ethernet
            autoconnect: yes
            ip:
```

```
      dhcp4: yes
      auto6: yes
    state: up
```

これらの設定では、**enp1s0** デバイスのイーサネット接続プロファイルを定義します。接続では、DHCP サーバーと IPv6 ステートレスアドレス自動設定 (SLAAC) から、IPv4 アドレス、IPv6 アドレス、デフォルトゲートウェイ、ルート、DNS サーバー、および検索ドメインを取得します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

8.4. NETWORK RHEL システムロールとデバイスパスを使用した動的 IP アドレスでのイーサネット接続設定

network RHEL システムロールを使用して、イーサネット接続をリモートで設定できます。動的 IP アドレス設定との接続の場合、NetworkManager は、DHCP サーバーから接続の IP 設定を要求します。

デバイスパスは、次のコマンドで識別できます。

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

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- サーバーに、物理または仮想のイーサネットデバイスが設定されている。
- DHCP サーバーをネットワークで使用できる。
- 管理対象ホストは、NetworkManager を使用してネットワークを設定します。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure an Ethernet connection with dynamic IP
      ansible.builtin.include_role:
        name: rhel-system-roles.network
      vars:
        network_connections:
          - name: example
            match:
              path:
                - pci-0000:00:0[1-3].0
                - &!pci-0000:00:02.0
            type: ethernet
            autoconnect: yes
            ip:
              dhcp4: yes
              auto6: yes
            state: up
```

これらの設定では、イーサネット接続プロファイルを定義します。接続では、DHCP サーバーと IPv6 ステートレスアドレス自動設定 (SLAAC) から、IPv4 アドレス、IPv6 アドレス、デフォルトゲートウェイ、ルート、DNS サーバー、および検索ドメインを取得します。

match パラメーターは、PCI ID **0000:00:0[1-3].0** に一致するデバイスには Ansible によってプレイを適用し、**0000:00:02.0** には適用しないことを定義します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

8.5. NETWORK RHEL システムロールを使用した VLAN タグ付けの設定

network RHEL システムロールを使用して、VLAN タグ付けを設定できます。この例では、イーサネット接続と、このイーサネット接続の上に ID **10** の VLAN を追加します。子デバイスの VLAN 接続には、IP、デフォルトゲートウェイ、および DNS の設定が含まれます。

環境に応じて、プレイを適宜調整します。以下に例を示します。

- ボンディングなどの他の接続でポートとして VLAN を使用する場合は、**ip** 属性を省略し、子設定で IP 設定を行います。

- VLAN でチーム、ブリッジ、またはボンディングデバイスを使用するには、**interface_name** と VLAN で使用するポートの **type** 属性を調整します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure a VLAN that uses an Ethernet connection
      ansible.builtin.include_role:
        name: rhel-system-roles.network
      vars:
        network_connections:
          # Add an Ethernet profile for the underlying device of the VLAN
          - name: enp1s0
            type: ethernet
            interface_name: enp1s0
            autoconnect: yes
            state: up
            ip:
              dhcp4: no
              auto6: no

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

これらの設定では、**enp1s0** デバイス上で動作する VLAN を定義します。VLAN インターフェイスの設定は以下のようになります。

- 静的 IPv4 アドレス: サブネットマスクが **/24** の **192.0.2.1**
- 静的 IPv6 アドレス - **2001:db8:1::1** (**/64** サブネットマスクあり)
- IPv4 デフォルトゲートウェイ - **192.0.2.254**
- IPv6 デフォルトゲートウェイ - **2001:db8:1::fffe**
- IPv4 DNS サーバー - **192.0.2.200**
- IPv6 DNS サーバー - **2001:db8:1::ffbb**
- DNS 検索ドメイン - **example.com**
- VLAN ID - **10**
  VLAN プロファイルの **parent** 属性は、**enp1s0** デバイス上で動作する VLAN を設定します。子デバイスの VLAN 接続には、IP、デフォルトゲートウェイ、および DNS の設定が含まれます。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

8.6. NETWORK RHEL システムロールを使用したネットワークブリッジの設定

network RHEL システムロールを使用して、ネットワークブリッジをリモートで設定できます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- サーバーに、2 つ以上の物理ネットワークデバイスまたは仮想ネットワークデバイスがインストールされている。

手順

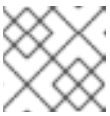
1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure a network bridge that uses two Ethernet ports
      ansible.builtin.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
```

これらの設定では、次の設定でネットワークブリッジを定義します。

- 静的 IPv4 アドレス: サブネットマスクが **/24** の **192.0.2.1**
- 静的 IPv6 アドレス - **2001:db8:1::1** (**/64** サブネットマスクあり)
- IPv4 デフォルトゲートウェイ - **192.0.2.254**
- IPv6 デフォルトゲートウェイ - **2001:db8:1::fffe**
- IPv4 DNS サーバー - **192.0.2.200**

- IPv6 DNS サーバー - **2001:db8:1::ffbb**
- DNS 検索ドメイン - **example.com**
- ブリッジのポート - **enp7s0** および **enp8s0**



注記

Linux ブリッジのポートではなく、ブリッジに IP 設定を指定します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

8.7. NETWORK RHEL システムロールを使用したネットワークボンディングの設定

network RHEL システムロールを使用して、ネットワークボンディングをリモートで設定できます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- サーバーに、2 つ以上の物理ネットワークデバイスまたは仮想ネットワークデバイスがインストールされている。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure a network bond that uses two Ethernet ports
```

```
    ansible.builtin.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
```

これらの設定では、次の設定を使用してネットワークボンディングを定義します。

- 静的 IPv4 アドレス: サブネットマスクが **/24** の **192.0.2.1**
- 静的 IPv6 アドレス - **2001:db8:1::1** (**/64** サブネットマスクあり)
- IPv4 デフォルトゲートウェイ - **192.0.2.254**
- IPv6 デフォルトゲートウェイ - **2001:db8:1::fffe**
- IPv4 DNS サーバー - **192.0.2.200**
- IPv6 DNS サーバー - **2001:db8:1::ffbb**
- DNS 検索ドメイン - **example.com**
- ボンディングのポート - **enp7s0** および **enp8s0**
- ボンディングモード - **active-backup**

注記

Linux ボンディングのポートではなく、ボンディングに IP 設定を設定します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

8.8. NETWORK RHEL システムロールを使用した IPOIB 接続の設定

network RHEL システムロールを使用して、IP over InfiniBand (IPoIB) デバイスの NetworkManager 接続プロファイルをリモートで作成できます。たとえば、Ansible Playbook を実行して、次の設定で **mlx4_ib0** インターフェイスの InfiniBand 接続をリモートで追加します。

- IPoIB デバイス - **mlx4_ib0.8002**
- パーティションキー **p_key** - **0x8002**
- 静的 **IPv4** アドレス - **192.0.2.1** と **/24** サブネットマスク
- 静的 **IPv6** アドレス - **2001:db8:1::1** と **/64** サブネットマスク

Ansible コントロールノードで以下の手順を実行します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- **mlx4_ib0** という名前の InfiniBand デバイスが管理対象ノードにインストールされている。
- 管理対象ノードが NetworkManager を使用してネットワークを設定している。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure IPoIB
      ansible.builtin.include_role:
        name: rhel-system-roles.network
      vars:
        network_connections:
          # InfiniBand connection mlx4_ib0
          - name: mlx4_ib0
            interface_name: mlx4_ib0
            type: infiniband

          # IPoIB device mlx4_ib0.8002 on top of mlx4_ib0
          - name: mlx4_ib0.8002
            type: infiniband
            autoconnect: yes
            infiniband:
              p_key: 0x8002
              transport_mode: datagram
            parent: mlx4_ib0
            ip:
              address:
                - 192.0.2.1/24
                - 2001:db8:1::1/64
            state: up
```

この例のように **p_key** パラメーターを設定する場合は、IPoIB デバイスで **interface_name** パラメーターを設定しないでください。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

1. **managed-node-01.example.com** ホストで、**mlx4_ib0.8002** デバイスの IP 設定を表示します。

```
# ip address show mlx4_ib0.8002
...
inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute ib0.8002
   valid_lft forever preferred_lft forever
inet6 2001:db8:1::1/64 scope link tentative noprefixroute
   valid_lft forever preferred_lft forever
```

2. **mlx4_ib0.8002** デバイスのパーティションキー (P_Key) を表示します。

```
# cat /sys/class/net/mlx4_ib0.8002/pkey
0x8002
```

3. **mlx4_ib0.8002** デバイスのモードを表示します。

```
# cat /sys/class/net/mlx4_ib0.8002/mode
datagram
```

関連情報

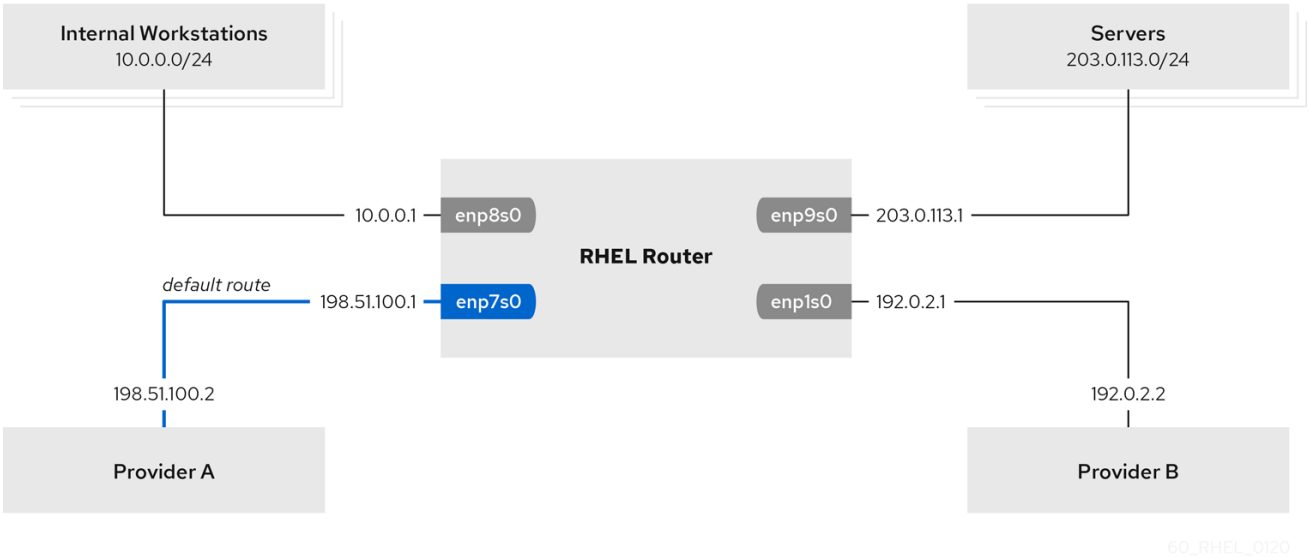
- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

8.9. NETWORK RHEL システムロールを使用した特定のサブネットから別のデフォルトゲートウェイへのトラフィックのルーティング

ポリシーベースのルーティングを使用して、特定のサブネットからのトラフィックに対して別のデフォルトゲートウェイを設定できます。たとえば、デフォルトルートを使用して、すべてのトラフィックをインターネットプロバイダー A にデフォルトでルーティングするルーターとして RHEL を設定できます。ただし、内部ワークステーションサブネットから受信したトラフィックはプロバイダー B にルーティングされます。

ポリシーベースのルーティングをリモートで複数のノードに設定するには、**network** RHEL システムロールを使用できます。

この手順では、次のネットワークトポロジーを想定しています。



前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

- 管理対象ノードは、**NetworkManager** および **firewalld** サービスを使用します。
- 設定する管理対象ノードには、次の 4 つのネットワークインターフェイスがあります。
  - **enp7s0** インターフェイスはプロバイダー A のネットワークに接続されます。プロバイダーのネットワークのゲートウェイ IP は **198.51.100.2** で、ネットワークは **/30** ネットワークマスクを使用します。
  - **enp1s0** インターフェイスはプロバイダー B のネットワークに接続されます。プロバイダーのネットワークのゲートウェイ IP は **192.0.2.2** で、ネットワークは **/30** ネットワークマスクを使用します。
  - **enp8s0** インターフェイスは、内部ワークステーションで **10.0.0.0/24** サブネットに接続されています。
  - **enp9s0** インターフェイスは、会社のサーバーで **203.0.113.0/24** サブネットに接続されています。
- 内部ワークステーションのサブネット内のホストは、デフォルトゲートウェイとして **10.0.0.1** を使用します。この手順では、この IP アドレスをルーターの **enp8s0** ネットワークインターフェイスに割り当てます。
- サーバーサブネット内のホストは、デフォルトゲートウェイとして **203.0.113.1** を使用します。この手順では、この IP アドレスをルーターの **enp9s0** ネットワークインターフェイスに割り当てます。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- name: Configuring policy-based routing
  hosts: managed-node-01.example.com
  tasks:
    - name: Routing traffic from a specific subnet to a different default gateway
      ansible.builtin.include_role:
        name: rhel-system-roles.network
      vars:
        network_connections:
          - name: Provider-A
            interface_name: enp7s0
            type: ethernet
            autoconnect: True
            ip:
              address:
                - 198.51.100.1/30
              gateway4: 198.51.100.2
              dns:
                - 198.51.100.200
            state: up
            zone: external

          - name: Provider-B
            interface_name: enp1s0
            type: ethernet
            autoconnect: True
```

```
      ip:
        address:
          - 192.0.2.1/30
        route:
          - network: 0.0.0.0
            prefix: 0
            gateway: 192.0.2.2
            table: 5000
      state: up
      zone: external

    - name: Internal-Workstations
      interface_name: enp8s0
      type: ethernet
      autoconnect: True
      ip:
        address:
          - 10.0.0.1/24
        route:
          - network: 10.0.0.0
            prefix: 24
            table: 5000
        routing_rule:
          - priority: 5
            from: 10.0.0.0/24
            table: 5000
      state: up
      zone: trusted

    - name: Servers
      interface_name: enp9s0
      type: ethernet
      autoconnect: True
      ip:
        address:
          - 203.0.113.1/24
      state: up
      zone: trusted
```

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

1. 内部ワークステーションサブネットの RHEL ホストで、以下を行います。
   a. **traceroute** パッケージをインストールします。

```
# dnf install traceroute
```

b. **traceroute** ユーティリティーを使用して、インターネット上のホストへのルートを表示します。

```
# traceroute redhat.com
traceroute to redhat.com (209.132.183.105), 30 hops max, 60 byte packets
 1  10.0.0.1 (10.0.0.1)     0.337 ms  0.260 ms  0.223 ms
 2  192.0.2.1 (192.0.2.1)   0.884 ms  1.066 ms  1.248 ms
 ...
```

コマンドの出力には、ルーターがプロバイダー B のネットワークである **192.0.2.1** 経由でパケットを送信することが表示されます。

2. サーバーのサブネットの RHEL ホストで、以下を行います。

a. **traceroute** パッケージをインストールします。

```
# dnf install traceroute
```

b. **traceroute** ユーティリティーを使用して、インターネット上のホストへのルートを表示します。

```
# traceroute redhat.com
traceroute to redhat.com (209.132.183.105), 30 hops max, 60 byte packets
 1  203.0.113.1 (203.0.113.1)    2.179 ms  2.073 ms  1.944 ms
 2  198.51.100.2 (198.51.100.2)  1.868 ms  1.798 ms  1.549 ms
 ...
```

コマンドの出力には、ルーターがプロバイダー A のネットワークである **198.51.100.2** 経由でパケットを送信することが表示されます。

3. RHEL システムロールを使用して設定した RHEL ルーターで、次の手順を実行します。

a. ルールのリストを表示します。

```
# ip rule list
0:      from all lookup local
5:    from 10.0.0.0/24 lookup 5000
32766:  from all lookup main
32767:  from all lookup default
```

デフォルトでは、RHEL には、**local** テーブル、**main** テーブル、および **default** テーブルのルールが含まれます。

b. テーブル **5000** のルートを表示します。

```
# ip route list table 5000
0.0.0.0/0 via 192.0.2.2 dev enp1s0 proto static metric 100
10.0.0.0/24 dev enp8s0 proto static scope link src 192.0.2.1 metric 102
```

c. インターフェイスとファイアウォールゾーンを表示します。

```
# firewall-cmd --get-active-zones
```

```
external
  interfaces: enp1s0 enp7s0
trusted
  interfaces: enp8s0 enp9s0
```

d. **external** ゾーンでマスカレードが有効になっていることを確認します。

```
# firewall-cmd --info-zone=external
external (active)
  target: default
  icmp-block-inversion: no
  interfaces: enp1s0 enp7s0
  sources:
  services: ssh
  ports:
  protocols:
  masquerade: yes
  ...
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

8.10. NETWORK RHEL システムロールを使用した 802.1X ネットワーク認証による静的イーサネット接続の設定

network RHEL システムロールを使用して、802.1X ネットワーク認証によるイーサネット接続をリモートで設定できます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- ネットワークは 802.1X ネットワーク認証をサポートしている。
- 管理対象ノードは NetworkManager を使用します。
- TLS 認証に必要な以下のファイルがコントロールノードにある。
  - クライアントキーは、**/srv/data/client.key** ファイルに保存されます。
  - クライアント証明書は **/srv/data/client.crt** ファイルに保存されます。
  - 認証局 (CA) 証明書は、**/srv/data/ca.crt** ファイルに保存されます。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- name: Configure an Ethernet connection with 802.1X authentication
  hosts: managed-node-01.example.com
  tasks:
    - name: Copy client key for 802.1X authentication
      ansible.builtin.copy:
        src: "/srv/data/client.key"
        dest: "/etc/pki/tls/private/client.key"
        mode: 0600

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

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

    - name: Configure connection
      ansible.builtin.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
```

これらの設定では、次の設定を使用して **enp1s0** デバイスのイーサネット接続プロファイルを定義します。

- 静的 IPv4 アドレス: サブネットマスクが **/24** の **192.0.2.1**
- 静的 IPv6 アドレス - **2001:db8:1::1** (**/64** サブネットマスクあり)

- IPv4 デフォルトゲートウェイ - **192.0.2.254**
- IPv6 デフォルトゲートウェイ - **2001:db8:1::fffe**
- IPv4 DNS サーバー - **192.0.2.200**
- IPv6 DNS サーバー - **2001:db8:1::ffbb**
- DNS 検索ドメイン - **example.com**
- **TLS** Extensible Authentication Protocol (EAP) を使用した 802.1X ネットワーク認証

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

8.11. NETWORK RHEL システムロールを使用した 802.1X ネットワーク認証による WI-FI 接続の設定

RHEL システムロールを使用すると、Wi-Fi 接続の作成を自動化できます。たとえば、Ansible Playbook を使用して、**wlp1s0** インターフェイスのワイヤレス接続プロファイルをリモートで追加できます。作成されたプロファイルは、802.1X 標準を使用して、wifi ネットワークに対してクライアントを認証します。Playbook は、DHCP を使用するように接続プロファイルを設定します。静的 IP 設定を設定するには、それに応じて **IP** ディクショナリーのパラメーターを調整します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- ネットワークは 802.1X ネットワーク認証をサポートしている。
- 管理対象ノードに **wpa_supplicant** パッケージをインストールしている。
- DHCP は、管理対象ノードのネットワークで使用できる。
- TLS 認証に必要な以下のファイルがコントロールノードにある。

- クライアントキーは、**/srv/data/client.key** ファイルに保存されます。
- クライアント証明書は **/srv/data/client.crt** ファイルに保存されます。
- CA 証明書は **/srv/data/ca.crt** ファイルに保存されます。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Configure a wifi connection with 802.1X authentication
  hosts: managed-node-01.example.com
  tasks:
    - name: Copy client key for 802.1X authentication
      ansible.builtin.copy:
        src: "/srv/data/client.key"
        dest: "/etc/pki/tls/private/client.key"
        mode: 0400

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

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

    - block:
        - ansible.builtin.import_role:
            name: rhel-system-roles.network
          vars:
            network_connections:
              - name: Configure the Example-wifi profile
                interface_name: wlp1s0
                state: up
                type: wireless
                autoconnect: yes
                ip:
                  dhcp4: true
                  auto6: true
                wireless:
                  ssid: "Example-wifi"
                  key_mgmt: "wpa-eap"
                ieee802_1x:
                  identity: "user_name"
                  eap: tls
                  private_key: "/etc/pki/tls/client.key"
                  private_key_password: "password"
                  private_key_password_flags: none
                  client_cert: "/etc/pki/tls/client.pem"
                  ca_cert: "/etc/pki/tls/cacert.pem"
                  domain_suffix_match: "example.com"
```

これらの設定では、**wlp1s0** インターフェイスの Wi-Fi 接続プロファイルを定義します。このプロファイルは、802.1X 標準を使用して、Wi-Fi ネットワークに対してクライアントを認証します。接続では、DHCP サーバーと IPv6 ステートレスアドレス自動設定 (SLAAC) から、IPv4 アドレス、IPv6 アドレス、デフォルトゲートウェイ、ルート、DNS サーバー、および検索ドメインを取得します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

8.12. NETWORK RHEL システムロールを使用して既存の接続にデフォルトゲートウェイを設定する

network RHEL システムロールを使用して、デフォルトゲートウェイを設定できます。

重要

network RHEL システムロールを使用するプレイの実行時に、プレイで指定した値と設定値が一致しない場合、当該ロールは同じ名前の既存の接続プロファイルをオーバーライドします。これらの値がデフォルトにリセットされないようにするには、IP 設定などの設定がすでに存在する場合でも、ネットワーク接続プロファイルの設定全体をプレイで必ず指定してください。

この手順では、すでに存在するかどうかに応じて、以下の設定で **enp1s0** 接続プロファイルを作成または更新します。

- 静的 IPv4 アドレス - **/24** サブネットマスクを持つ **198.51.100.20**
- 静的 IPv6 アドレス - **2001:db8:1::1** (**/64** サブネットマスクあり)
- IPv4 デフォルトゲートウェイ - **198.51.100.254**
- IPv6 デフォルトゲートウェイ - **2001:db8:1::fffe**
- IPv4 DNS サーバー - **198.51.100.200**
- IPv6 DNS サーバー - **2001:db8:1::ffbb**
- DNS 検索ドメイン - **example.com**

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure an Ethernet connection with static IP and default gateway
      ansible.builtin.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
```

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

8.13. NETWORK RHEL システムロールを使用した静的ルートの設定

network RHEL システムロールを使用して、静的ルートを設定できます。

重要

network RHEL システムロールを使用するプレイの実行時に、プレイで指定した値と設定値が一致しない場合、当該ロールは同じ名前の既存の接続プロファイルをオーバーライドします。これらの値がデフォルトにリセットされないようにするには、IP 設定などの設定がすでに存在する場合でも、ネットワーク接続プロファイルの設定全体をプレイで必ず指定してください。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure an Ethernet connection with static IP and additional routes
      ansible.builtin.include_role:
        name: rhel-system-roles.network
      vars:
        network_connections:
          - name: enp7s0
            type: ethernet
            autoconnect: yes
            ip:
              address:
                - 192.0.2.1/24
                - 2001:db8:1::1/64
              gateway4: 192.0.2.254
              gateway6: 2001:db8:1::fffe
              dns:
                - 192.0.2.200
                - 2001:db8:1::ffbb
              dns_search:
                - example.com
              route:
                - network: 198.51.100.0
                  prefix: 24
                  gateway: 192.0.2.10
                - network: 2001:db8:2::
                  prefix: 64
                  gateway: 2001:db8:1::10
            state: up
```

この手順では、すでに存在するかどうかに応じて、以下の設定で **enp7s0** 接続プロファイルを作成または更新します。

- 静的 IPv4 アドレス: サブネットマスクが **/24** の **192.0.2.1**
- 静的 IPv6 アドレス - **2001:db8:1::1** (**/64** サブネットマスクあり)
- IPv4 デフォルトゲートウェイ - **192.0.2.254**
- IPv6 デフォルトゲートウェイ - **2001:db8:1::fffe**
- IPv4 DNS サーバー - **192.0.2.200**
- IPv6 DNS サーバー - **2001:db8:1::ffbb**
- DNS 検索ドメイン - **example.com**
- 静的ルート:
  - **198.51.100.0/24** のゲートウェイ **192.0.2.10**
  - **2001:db8:2::/64** とゲートウェイ **2001:db8:1::10**

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

1. 管理対象ノードで以下を行います。

   a. IPv4 ルートを表示します。

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

   b. IPv6 ルートを表示します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

8.14. NETWORK RHEL システムロールを使用した ETHTOOL オフロード機能の設定

network RHEL システムロールを使用して、NetworkManager 接続の **ethtool** 機能を設定できます。

重要

network RHEL システムロールを使用するプレイの実行時に、プレイで指定した値と設定値が一致しない場合、当該ロールは同じ名前の既存の接続プロファイルをオーバーライドします。これらの値がデフォルトにリセットされないようにするには、IP 設定などの設定がすでに存在する場合でも、ネットワーク接続プロファイルの設定全体をプレイで必ず指定してください。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure an Ethernet connection with ethtool features
      ansible.builtin.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:
              features:
                gro: "no"
                gso: "yes"
                tx_sctp_segmentation: "no"
            state: up
```

この Playbook は、**enp1s0** 接続プロファイルを次の設定で作成します。プロファイルがすでに存在する場合は、次の設定に更新します。

- 静的 **IPv4** アドレス - **/24** サブネットマスクを持つ **198.51.100.20**
- 静的 **IPv6** アドレス - **2001:db8:1::1** と **/64** サブネットマスク
- **IPv4** デフォルトゲートウェイ - **198.51.100.254**
- **IPv6** デフォルトゲートウェイ - **2001:db8:1::fffe**
- **IPv4** DNS サーバー - **198.51.100.200**
- **IPv6** DNS サーバー - **2001:db8:1::ffbb**
- DNS 検索ドメイン - **example.com**
- **ethtool** 機能:
  - 汎用受信オフロード (GRO): 無効
  - Generic segmentation offload(GSO): 有効化
  - TX stream control transmission protocol (SCTP) segmentation: 無効

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

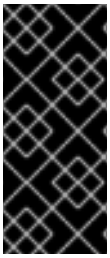
```
$ ansible-playbook ~/playbook.yml
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

8.15. NETWORK RHEL システムロールを使用した ETHTOOL COALESCE の設定

network RHEL システムロールを使用して、NetworkManager 接続の **ethtool** coalesce を設定できます。



重要

network RHEL システムロールを使用するプレイの実行時に、プレイで指定した値と設定値が一致しない場合、当該ロールは同じ名前の既存の接続プロファイルをオーバーライドします。これらの値がデフォルトにリセットされないようにするには、IP 設定などの設定がすでに存在する場合でも、ネットワーク接続プロファイルの設定全体をプレイで必ず指定してください。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure an Ethernet connection with ethtool coalesce settings
      ansible.builtin.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:
              coalesce:
                rx_frames: 128
                tx_frames: 128
            state: up
```

この Playbook は、**enp1s0** 接続プロファイルを次の設定で作成します。プロファイルがすでに存在する場合は、次の設定に更新します。

- 静的 IPv4 アドレス - /**24** サブネットマスクを持つ **198.51.100.20**
- 静的 IPv6 アドレス - **2001:db8:1::1** (/**64** サブネットマスクあり)
- IPv4 デフォルトゲートウェイ - **198.51.100.254**
- IPv6 デフォルトゲートウェイ - **2001:db8:1::fffe**
- IPv4 DNS サーバー - **198.51.100.200**
- IPv6 DNS サーバー - **2001:db8:1::ffbb**

- DNS 検索ドメイン - **example.com**
- **ethtool** coalesce の設定:
  - RX フレーム: **128**
  - TX フレーム: **128**

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

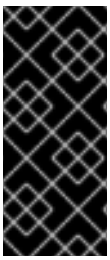
8.16. NETWORK RHEL システムロールを使用して、高いパケットドロップ率を減らすためにリングバッファーサイズを増やす

パケットドロップ率が原因でアプリケーションがデータの損失、タイムアウト、またはその他の問題を報告する場合は、イーサネットデバイスのリングバッファーのサイズを増やします。

リングバッファーは循環バッファーであり、オーバーフローによって既存のデータが上書きされます。ネットワークカードは、送信 (TX) および受信 (RX) リングバッファーを割り当てます。受信リングバッファーは、デバイスドライバーとネットワークインターフェイスコントローラー (NIC) の間で共有されます。データは、ハードウェア割り込みまたは SoftIRQ とも呼ばれるソフトウェア割り込みによって NIC からカーネルに移動できます。

カーネルは RX リングバッファーを使用して、デバイスドライバーが着信パケットを処理できるようになるまで着信パケットを格納します。デバイスドライバーは、通常は SoftIRQ を使用して RX リングをドレインします。これにより、着信パケットは **sk_buff** または **skb** と呼ばれるカーネルデータ構造に配置され、カーネルを経由して関連するソケットを所有するアプリケーションまでの移動を開始します。

カーネルは TX リングバッファーを使用して、ネットワークに送信する必要がある発信パケットを保持します。これらのリングバッファーはスタックの一番下にあり、パケットドロップが発生する重要なポイントであり、ネットワークパフォーマンスに悪影響を及ぼします。



重要

network RHEL システムロールを使用するプレイの実行時に、プレイで指定した値と設定値が一致しない場合、当該ロールは同じ名前の既存の接続プロファイルをオーバーライドします。これらの値がデフォルトにリセットされないようにするには、IP 設定などの設定がすでに存在する場合でも、ネットワーク接続プロファイルの設定全体をプレイで必ず指定してください。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- デバイスがサポートする最大リングバッファーサイズを把握している。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure an Ethernet connection with increased ring buffer sizes
      ansible.builtin.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:
              ring:
                rx: 4096
                tx: 4096
            state: up
```

この Playbook は、**enp1s0** 接続プロファイルを次の設定で作成します。プロファイルがすでに存在する場合は、次の設定に更新します。

- 静的 **IPv4** アドレス - **/24** サブネットマスクを持つ **198.51.100.20**
- 静的 **IPv6** アドレス - **2001:db8:1::1** と **/64** サブネットマスク
- **IPv4** デフォルトゲートウェイ - **198.51.100.254**
- **IPv6** デフォルトゲートウェイ - **2001:db8:1::fffe**
- **IPv4** DNS サーバー - **198.51.100.200**

- **IPv6** DNS サーバー - **2001:db8:1::ffbb**
- DNS 検索ドメイン - **example.com**
- リングバッファーエントリーの最大数:
  - 受信 (RX): 4096
  - 送信 (TX): 4096

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

8.17. NETWORK RHEL システムロールのネットワーク状態

network RHEL システムロールは、Playbook でデバイスを設定するための状態設定をサポートしています。これには、**network_state** 変数の後に状態設定を使用します。

Playbook で **network_state** 変数を使用する利点:

- 状態設定で宣言型の方法を使用すると、インターフェイスを設定でき、NetworkManager はこれらのインターフェイスのプロファイルをバックグラウンドで作成します。
- **network_state** 変数を使用すると、変更が必要なオプションを指定できます。他のすべてのオプションはそのまま残ります。ただし、**network_connections** 変数を使用して、ネットワーク接続プロファイルを変更するには、すべての設定を指定する必要があります。

たとえば、動的 IP アドレス設定でイーサネット接続を作成するには、Playbook で次の **vars** ブロックを使用します。

状態設定を含む Playbook	通常の Playbook

```
vars:
  network_state:
    interfaces:
    - name: enp7s0
      type: ethernet
      state: up
      ipv4:
        enabled: true
        auto-dns: true
        auto-gateway: true
        auto-routes: true
        dhcp: true
      ipv6:
        enabled: true
        auto-dns: true
        auto-gateway: true
        auto-routes: true
        autoconf: true
        dhcp: true
```

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

たとえば、上記のように作成した動的 IP アドレス設定の接続ステータスのみを変更するには、Playbook で次の **vars** ブロックを使用します。

状態設定を含む Playbook

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

通常の Playbook

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

第9章 RHEL システムロールを使用した FIREWALLD の設定

firewall RHEL システムロールを使用すると、一度に複数のクライアントに **firewalld** サービスを設定できます。この解決策は以下のとおりです。

- 入力設定が効率的なインターフェイスを提供する。
- 目的の **firewalld** パラメーターを 1 か所で保持する。

コントロールノードで **firewall** ロールを実行すると、RHEL システムロールが **firewalld** パラメーターを管理対象ノードに即座に適用し、再起動後もパラメーターを維持します。

9.1. RHEL システムロール FIREWALL の概要

RHEL システムロールは、Ansible 自動化ユーティリティーのコンテンツセットです。このコンテンツは、Ansible 自動化ユーティリティーとともに、複数のシステムをリモートで管理するための一貫した設定インターフェイスを提供します。

RHEL システムロールの **rhel-system-roles.firewall** ロールが、**firewalld** サービスの自動設定のために導入されました。**rhel-system-roles** パッケージに、この RHEL システムロールとリファレンスドキュメントが含まれています。

1 つ以上のシステムに **firewalld** パラメーターを自動的に適用するには、Playbook で **firewall** RHEL システムロール変数を使用します。Playbook は、テキストベースの YAML 形式で記述された 1 つ以上のプレイのリストです。

インベントリーファイルを使用して、Ansible が設定するシステムセットを定義できます。

firewall ロールを使用すると、以下のような異なる **firewalld** パラメーターを設定できます。

- ゾーン。
- パケットが許可されるサービス。
- ポートへのトラフィックアクセスの付与、拒否、または削除。
- ゾーンのポートまたはポート範囲の転送。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.firewall/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/firewall/** ディレクトリー
- Playbook の使用
- インベントリーの構築方法

9.2. RHEL システムロールを使用した FIREWALLD 設定のリセット

firewall RHEL システムロールを使用すると、**firewalld** 設定をデフォルトの状態にリセットできます。変数リストに **previous:replaced** パラメーターを追加すると、RHEL システムロールが既存のユーザー定義の設定をすべて削除し、**firewalld** をデフォルトにリセットします。**previous:replaced** パラメーターを他の設定と組み合わせると、**firewall** ロールは新しい設定を適用する前に既存の設定をすべて削除します。

Ansible コントロールノードで以下の手順を実行します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

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

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

- 管理対象ノードで **root** として次のコマンドを実行し、すべてのゾーンを確認します。

```
# firewall-cmd --list-all-zones
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.firewall/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/firewall/** ディレクトリー

9.3. RHEL システムロールを使用して、FIREWALLD の着信トラフィックをあるローカルポートから別のローカルポートに転送する

firewall ロールを使用すると、複数の管理対象ホストで設定が永続化されるので **firewalld** パラメーターをリモートで設定できます。

Ansible コントロールノードで以下の手順を実行します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

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

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

- 管理対象ホストで、**firewalld** 設定を表示します。

```
# firewall-cmd --list-forward-ports
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.firewall/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/firewall/** ディレクトリー

9.4. RHEL システムロールを使用した FIREWALLD のポートの管理

firewall RHEL システムロールを使用して、着信トラフィック用のローカルファイアウォールのポートを開閉し、再起動後も新しい設定を維持できます。たとえば、HTTPS サービスの着信トラフィックを許可するようにデフォルトゾーンを設定できます。

Ansible コントロールノードで以下の手順を実行します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Configure firewalld
  hosts: managed-node-01.example.com
  tasks:
    - name: Allow incoming HTTPS traffic to the local host
      ansible.builtin.include_role:
        name: rhel-system-roles.firewall
      vars:
        firewall:
          - port: 443/tcp
            service: http
            state: enabled
            runtime: true
            permanent: true
```

permanent: true オプションを使用すると、再起動後も新しい設定が維持されます。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

- 管理対象ノードで、**HTTPS** サービスに関連付けられた **443/tcp** ポートが開いていることを確認します。

```
# firewall-cmd --list-ports
443/tcp
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.firewall/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/firewall/** ディレクトリー

9.5. RHEL システムロールを使用した FIREWALLD DMZ ゾーンの設定

システム管理者は、**firewall** RHEL システムロールを使用して、**enp1s0** インターフェイス上に **dmz** ゾーンを設定し、ゾーンへの **HTTPS** トラフィックを許可できます。これにより、外部ユーザーが Web サーバーにアクセスできるようにします。

Ansible コントロールノードで以下の手順を実行します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- name: Configure firewalld
  hosts: managed-node-01.example.com
  tasks:
    - name: Creating a DMZ with access to HTTPS port and masquerading for hosts in DMZ
      ansible.builtin.include_role:
        name: rhel-system-roles.firewall
      vars:
        firewall:
          - zone: dmz
            interface: enp1s0
            service: https
            state: enabled
            runtime: true
            permanent: true
```

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

検証

- 管理ノードで、**dmz** ゾーンに関する詳細情報を表示します。

```
# firewall-cmd --zone=dmz --list-all
dmz (active)
  target: default
  icmp-block-inversion: no
  interfaces: enp1s0
  sources:
  services: https ssh
  ports:
  protocols:
  forward: no
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.firewall/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/firewall/** ディレクトリー

第10章 RHEL システムロールを使用した POSTFIX MTA の設定

postfix RHEL システムロールを使用すると、Postfix サービスの自動設定を一貫して効率化できます。Postfix サービスは、モジュラー設計およびさまざまな設定オプションを備えた Sendmail 互換のメール転送エージェント (MTA) です。**rhel-system-roles** パッケージに、この RHEL システムロールとリファレンスドキュメントが含まれています。

10.1. POSTFIX システムロールを使用した基本的な POSTFIX MTA 管理の自動化

postfix RHEL システムロールを使用して、管理対象ノードに Postfix Mail Transfer Agent をインストール、設定、起動できます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Manage postfix
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.postfix
  vars:
    postfix_conf:
      relay_domains: $mydestination
        relayhost: example.com
```

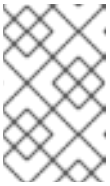
- **gethostname()** 関数によって返される完全修飾ドメイン名 (FQDN) とは異なるホスト名を Postfix で使用する場合は、ファイルの **postfix_conf:** 行の下に **myhostname** パラメーターを追加します。

```
myhostname = smtp.example.com
```

- ドメイン名が **myhostname** パラメーターのドメイン名と異なる場合は、**mydomain** パラメーターを追加します。それ以外の場合は、**$myhostname** から最初のコンポーネントを除いた値が使用されます。

```
mydomain = <example.com>
```

- **postfix_manage_firewall: true** 変数を使用して、サーバー上のファイアウォールで SMTP ポートを開きます。
  SMTP 関連のポートである **25/tcp**、**465/tcp**、および **587/tcp** を管理します。変数が **false** に設定されている場合、**postfix** ロールはファイアウォールを管理しません。デフォルトは **false** です。



注記

postfix_manage_firewall 変数の用途はポートの追加に限定されています。ポートの削除には使用できません。ポートを削除する場合は、**firewall** RHEL システムロールを直接使用します。

- 標準以外のポートを使用する場合は、**postfix_manage_selinux: true** 変数を設定して、ポートがサーバー上で SELinux 用に適切にラベル付けされるようにします。

注記

postfix_manage_selinux 変数の用途は、SELinux ポリシーへのルールの追加に限定されています。ポリシーからルールを削除することはできません。ルールを削除する場合は、**selinux** RHEL システムロールを直接使用します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.postfix/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/postfix/** ディレクトリー

10.2. POSTFIX RHEL システムロールの変数

postfix RHEL システムロールの変数を使用して、Postfix Mail Transfer Agent (MTA) の設定をカスタマイズできます。

基本的な設定には次の変数を使用します。その他の変数については、**rhel-system-roles** パッケージとともにインストールされるドキュメントを参照してください。

postfix_conf

この変数を使用して、サポートされているすべての **postfix** 設定パラメーターのキーまたは値のペアを追加します。デフォルトでは、**postfix_conf** には値が設定されていません。

```
postfix_conf:
  relayhost: example.com
```

previous: replaced

この変数を使用して、既存の設定を削除し、クリーンな **postfix** インストールに目的の設定を適用します。

```
postfix_conf:
  relayhost: example.com
  previous: replaced
```

postfix_check

この変数を使用して、**postfix** ロールを起動する前に、設定変更を検証するための確認が実行されたかどうかを判断します。デフォルト値は **true** です。
以下に例を示します。

```
postfix_check: true
```

postfix_backup

この変数を **true** に設定して、設定のバックアップコピーを 1 つ作成します。デフォルト値は **false** です。
以前のバックアップを上書きする場合は、以下のコマンドを入力します。

```
# cp /etc/postfix/main.cf /etc/postfix/main.cf.backup
```

postfix_backup の値を **true** に変更した場合は、**postfix_backup_multiple** の値も false に設定する必要があります。

```
postfix_backup: true
postfix_backup_multiple: false
```

postfix_backup_multiple

この変数を **true** に設定して、タイムスタンプ付きの設定のバックアップコピーを作成します。デフォルト値は **true** です。
複数のバックアップコピーを保持するには、次のコマンドを入力します。

```
# cp /etc/postfix/main.cf /etc/postfix/main.cf.$(date -Isec)
```

postfix_backup_multiple:true 設定は **postfix_backup** をオーバーライドします。**postfix_backup** を使用する場合は、**postfix_backup_multiple:false** を設定する必要があります。

postfix_manage_firewall

この変数を使用して、**postfix** ロールを **firewall** ロールと統合し、ポートアクセスを管理します。デフォルトでは、変数は **false** に設定されます。**postfix** ロールからポートアクセスを自動的に管理する場合は、変数を **true** に設定します。

postfix_manage_selinux

この変数を使用して、**postfix** ロールを **selinux** ロールと統合し、ポートアクセスを管理します。デフォルトでは、変数は **false** に設定されます。**postfix** ロールからポートアクセスを自動的に管理する場合は、変数を **true** に設定します。

第11章 システムロールを使用した SELINUX の設定

selinux RHEL システムロールを使用して、他のシステムで SELinux 権限を設定および管理できます。

11.1. SELINUX システムロールの概要

RHEL システムロールは、複数の RHEL システムをリモートで管理するための一貫した設定インターフェイスを提供する Ansible ロールおよびモジュールのコレクションです。**selinux** RHEL システムロールを使用すると、次のアクションを実行できます。

- SELinux ブール値、ファイルコンテキスト、ポート、およびログインに関連するローカルポリシーの変更を消去します。
- SELinux ポリシーブール値、ファイルコンテキスト、ポート、およびログインの設定
- 指定されたファイルまたはディレクトリーでファイルコンテキストを復元します。
- SELinux モジュールの管理

次の表は、**selinux** RHEL システムロールで使用可能な入力変数の概要を示しています。

表11.1 selinux システムロールの変数

ロール変数	説明	CLI の代替手段
selinux_policy	ターゲットプロセスまたは複数レベルのセキュリティー保護を保護するポリシーを選択します。	**/etc/selinux/config** の **SELINUXTYPE**
selinux_state	SELinux モードを切り替えます。	**/etc/selinux/config** の **setenforce** and **SELINUX**
selinux_booleans	SELinux ブール値を有効または無効にします。	**setsebool**
selinux_fcontexts	SELinux ファイルコンテキストマッピングを追加または削除します。	**semanage fcontext**
selinux_restore_dirs	ファイルシステムツリー内の SELinux ラベルを復元します。	**restorecon -R**
selinux_ports	ポートに SELinux ラベルを設定します。	**semanage port**
selinux_logins	ユーザーを SELinux ユーザーマッピングに設定します。	**semanage login**
selinux_modules	SELinux モジュールのインストール、有効化、無効化、または削除を行います。	**semodule**

rhel-system-roles パッケージによりインストールされる **/usr/share/doc/rhel-system-roles/selinux/example-selinux-playbook.yml** のサンプル Playbook は、Enforcing モードでターゲットポリシーを設定する方法を示しています。Playbook は、複数のローカルポリシーの変更を適用し、**tmp/test_dir/** ディレクトリーのファイルコンテキストを復元します。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.selinux/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/selinux/** ディレクトリー

11.2. SELINUX システムロールを使用して複数のシステムに SELINUX 設定を適用する

selinux RHEL システムロールを使用すると、検証済みの SELinux 設定を使用して Ansible Playbook を準備および適用できます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. Playbook を準備します。ゼロから準備するか、**rhel-system-roles** パッケージの一部としてインストールされたサンプル Playbook を変更してください。

```
# cp /usr/share/doc/rhel-system-roles/selinux/example-selinux-playbook.yml <my-selinux-playbook.yml>
# vi <my-selinux-playbook.yml>
```

2. シナリオに合わせて Playbook の内容を変更します。たとえば、次の部分では、システムが SELinux モジュール **selinux-local-1.pp** をインストールして有効にします。

```
selinux_modules:
- { path: "selinux-local-1.pp", priority: "400" }
```

3. 変更を保存し、テキストエディターを終了します。
4. Playbook の構文を検証します。

```
# ansible-playbook <my-selinux-playbook.yml> --syntax-check
```

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

5. Playbook を実行します。

```
# ansible-playbook <my-selinux-playbook.yml>
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.selinux/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/selinux/** ディレクトリー
- ナレッジベース記事 SELinux hardening with Ansible

11.3. SELINUX RHEL システムロールを使用したポートの管理

selinux RHEL システムロールを使用すると、複数のシステムにわたる一貫した SELinux でのポートアクセス管理を自動化できます。これは、たとえば、Apache HTTP サーバーを別のポートでリッスンするように設定する場合に役立ちます。これを実行するには、特定のポート番号に **http_port_t** SELinux タイプを割り当てる **selinux** RHEL システムロールを使用して Playbook を作成します。管理対象ノードで Playbook を実行すると、SELinux ポリシーで定義された特定のサービスがこのポートにアクセスできるようになります。

SELinux でのポートアクセス管理は、**seport** モジュール (ロール全体を使用するよりも高速) を使用するか、**selinux** RHEL システムロール (SELinux 設定で他の変更も行う場合に便利) を使用することで自動化できます。これらの方法は同等です。実際、**selinux** RHEL システムロールは、ポートを設定するときに **seport** モジュールを使用します。どちらの方法も、管理対象ノードでコマンド **semanage port -a -t http_port_t -p tcp <port_number>** を入力するのと同じ効果があります。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- オプション: **semanage** コマンドを使用してポートの状態を確認するには、**policycoreutils-python-utils** パッケージをインストールする必要があります。

手順

- 他の変更を加えずにポート番号だけを設定するには、**seport** モジュールを使用します。

```
- name: Allow Apache to listen on tcp port <port_number>
  community.general.seport:
    ports: <port_number>
    proto: tcp
    setype: http_port_t
    state: present
```

  <port_number> は、**http_port_t** タイプを割り当てるポート番号に置き換えます。

- SELinux のその他のカスタマイズを伴うより複雑な設定を管理対象ノードに行う場合は、**selinux** RHEL システムロールを使用します。Playbook ファイル (例: **~/playbook.yml**) を作成し、次の内容を追加します。

```
---
- name: Modify SELinux port mapping example
  hosts: all
```

```
vars:
  # Map tcp port <port_number> to the 'http_port_t' SELinux port type
  selinux_ports:
    - ports: <port_number>
      proto: tcp
      setype: http_port_t
      state: present

tasks:
  - name: Include selinux role
    ansible.builtin.include_role:
      name: rhel-system-roles.selinux
```

<port_number> は、**http_port_t** タイプを割り当てるポート番号に置き換えます。

検証

- ポートが **http_port_t** タイプに割り当てられていることを確認します。

```
# semanage port --list | grep http_port_t
http_port_t                    tcp   <port_number>, 80, 81, 443, 488, 8008, 8009, 8443, 9000
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.selinux/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/selinux/** ディレクトリー

第12章 SYSTEMD RHEL システムロールを使用した SYSTEMD ユニットの管理

systemd RHEL システムロールを使用すると、Red Hat Ansible Automation Platform を使用してユニットファイルをデプロイし、複数のシステム上で **systemd** ユニットを管理できます。

systemd RHEL システムロール Playbook で **systemd_units** 変数を使用すると、ターゲットシステム上の **systemd** ユニットのステータスを把握できます。この変数はディクショナリーのリストを表示します。各ディクショナリーエントリーは、マネージドホスト上に存在する 1 つの **systemd** ユニットの状態と設定を記述します。**systemd_units** 変数は、タスク実行の最終ステップとして更新され、ロールがすべてのタスクを実行した後の状態をキャプチャーします。

12.1. SYSTEMD RHEL システムロールの変数

systemd RHEL システムロールに次の入力変数を設定することで、 **systemd** システムとサービスマネージャーの動作をカスタマイズできます。

systemd_unit_files

ターゲットホストにデプロイする **systemd** ユニットファイル名のリストを指定します。

systemd_unit-file_templates

テンプレートとして扱う必要がある systemd ユニットファイル名のリストを指定します。それぞれの名前は、Jinja テンプレートファイルに対応している必要があります。たとえば、**<name>.service** ユニットファイルの場合、**<name>.service** Jinja テンプレートファイルまたは **<name>.service.j2** Jinja テンプレートファイルのいずれかを使用できます。ローカルテンプレートファイルに **.j2** 接尾辞が付いている場合、Ansible は最終的なユニットファイル名を作成する前に接尾辞を削除します。

systemd_dropins

systemd ドロップイン設定ファイルのリストを指定して、ユニットファイルに直接変更を加えずに **systemd** ユニットの動作を変更または拡張します。
Playbook で **systemd_dropins = <name>.service.conf** 変数を設定すると、Ansible はローカルの **<name>.service.conf** ファイルを取得し、管理対象ノード上に常に **99-override.conf** という名前のドロップインファイルを作成し、このドロップインファイルを使用して、**<name>.service** **systemd** ユニットを変更します。

systemd_started_units

systemd が起動するユニット名のリストを指定します。

systemd_stopped_units

この変数を使用して、**systemd** が停止する必要があるユニット名のリストを指定します。

systemd_restarted_units

systemd が再起動する必要があるユニット名のリストを指定します。

systemd_reloaded_units

systemd がリロードする必要があるユニット名のリストを指定します。

systemd_enabled_units

systemd が有効にする必要があるユニット名のリストを指定します。

systemd_disabled_units

systemd が無効にする必要があるユニット名のリストを指定します。

systemd_masked_units

systemd がマスクする必要があるユニット名のリストを指定します。

systemd_unmasked_units

systemd がマスクを解除する必要があるユニット名のリストを指定します。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.systemd/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/systemd/** ディレクトリー

12.2. SYSTEMD システムロールを使用した SYSTEMD ユニットのデプロイと起動

systemd RHEL システムロールを適用して、ターゲットホスト上で **systemd** ユニット管理に関連するタスクを実行できます。Playbook で **systemd** RHEL システムロール変数を設定して、**systemd** がどのユニットファイルを管理し、起動し、有効にするかを定義します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- name: Deploy and start systemd unit
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.systemd
  vars:
    systemd_unit_files:
      - <name1>.service
      - <name2>.service
      - <name3>.service
    systemd_started_units:
      - <name1>.service
      - <name2>.service
      - <name3>.service
    systemd_enabled_units:
      - <name1>.service
      - <name2>.service
      - <name3>.service
```

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.systemd/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/systemd/** ディレクトリー

第13章 RHEL システムロールを使用したロギングの設定

システム管理者は、**logging** RHEL システムロールを使用して、RHEL ホストをロギングサーバーとして設定し、多くのクライアントシステムからログを収集できます。

13.1. LOGGING システムロール

logging RHEL システムロールを使用すると、ローカルホストとリモートホストにロギング設定をデプロイできます。

ロギングソリューションは、ログと複数のロギング出力を読み取る複数の方法を提供します。

たとえば、ロギングシステムは以下の入力を受け取ることができます。

- ローカルファイル
- **systemd/journal**
- ネットワーク上の別のロギングシステム

さらに、ロギングシステムでは以下を出力できます。

- **/var/log** ディレクトリーのローカルファイルに保存されているログ
- Elasticsearch に送信されたログ
- 別のロギングシステムに転送されたログ

logging RHEL システムロールを使用すると、状況に合わせて入力と出力を組み合わせることができます。たとえば、**journal** からの入力をローカルのファイルに保存しつつも、複数のファイルから読み込んだ入力を別のロギングシステムに転送してそのローカルのログファイルに保存するようにロギングソリューションを設定できます。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/logging/** ディレクトリー
- RHEL システムロール

13.2. LOGGING システムロールの変数

logging RHEL システムロール Playbook では、**logging_inputs** パラメーターで入力を定義し、**logging_outputs** パラメーターで出力を定義し、**logging_flows** パラメーターで入力と出力の関係を定義します。**logging** RHEL システムロールは、ロギングシステムを設定するための追加オプションを使用してこれらの変数を処理します。暗号化や自動ポート管理を有効にすることもできます。

注記

現在、**logging** RHEL システムロールで使用できるロギングシステムは **Rsyslog** だけです。

- **logging_inputs**: ロギングソリューションの入力リスト。

- **name**: 入力の一意の名前。**logging_flows** での使用: 入力リストおよび生成された **config** ファイル名の一部で使用されます。
- **type**: 入力要素のタイプ。type は、**roles/rsyslog/{tasks,vars}/inputs/** のディレクトリー名に対応するタスクタイプを指定します。
  - **basics**: **systemd** ジャーナルまたは **unix** ソケットからの入力を設定する入力。
    - **kernel_message**: **true** に設定されている場合に **imklog** を読み込みます。デフォルトは **false** です。
    - **use_imuxsock**: **imjournal** ではなく **imuxsock** を使用します。デフォルトは **false** です。
    - **ratelimit_burst**: **ratelimit_interval** 内に出力できるメッセージの最大数。**use_imuxsock** が false の場合、デフォルトで **20000** に設定されます。**use_imuxsock** が true の場合、デフォルトで **200** に設定されます。
    - **ratelimit_interval**: **ratelimit_burst** を評価する間隔。**use_imuxsock** が false の場合、デフォルトで 600 秒に設定されます。**use_imuxsock** が true の場合、デフォルトで 0 に設定されます。0 はレート制限がオフであることを示します。
    - **persist_state_interval**: ジャーナルの状態は、**value** メッセージごとに永続化されます。デフォルトは **10** です。**use_imuxsock** が false の場合のみ、有効です。
  - **files**: ローカルファイルからの入力を設定する入力。
  - **Remote**: ネットワークを介して他のロギングシステムからの入力を設定する入力。
- **状態**: 設定ファイルの状態。**present** または **absent**。デフォルトは **present** です。

- **logging_outputs**: ロギングソリューションの出力リスト。
  - **files**: ローカルファイルへの出力を設定する出力。
  - **forwards**: 別のロギングシステムへの出力を設定する出力。
  - **remote_files**: 別のロギングシステムからの出力をローカルファイルに設定する出力。
- **logging_flows**: **logging_inputs** および **logging_outputs** の関係を定義するフローのリスト。**logging_flows** 変数には以下が含まれます。
  - **name**: フローの一意の名前。
  - **inputs**: **logging_inputs** 名の値のリスト。
  - **outputs**: **logging_outputs** 名の値のリスト。
- **logging_manage_firewall**: **true** に設定すると、**logging** RHEL システムロールが **firewall** RHEL システムロールを使用してポートアクセスを自動的に管理します。
- **logging_manage_selinux**: **true** に設定すると、**logging** RHEL システムロールが **selinux** RHEL システムロールを使用してポートアクセスを自動的に管理します。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** ファイル

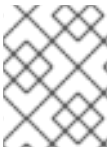
- **/usr/share/doc/rhel-system-roles/logging/** ディレクトリー

13.3. ローカルの LOGGING システムロールの適用

Ansible Playbook を準備して適用し、別のマシンにロギングソリューションを設定します。各マシンはログをローカルに記録します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。



注記

rsyslog パッケージをインストールする必要はありません。RHEL システムロールがデプロイ時に **rsyslog** をインストールするためです。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- name: Deploying basics input and implicit files output
  hosts: managed-node-01.example.com
  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. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

1. **/etc/rsyslog.conf** ファイルの構文をテストします。

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

2. システムがログにメッセージを送信していることを確認します。

   a. テストメッセージを送信します。

```
# logger test
```

   b. **/var/log/messages ログ** を表示します。以下に例を示します。

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

   <hostname> はクライアントシステムのホスト名に置き換えます。ログには、logger コマンドを入力したユーザーのユーザー名 (この場合は **root**) が含まれていることに注意してください。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/logging/** ディレクトリー

13.4. ローカルの LOGGING システムロールでログをフィルタリングする

rsyslog プロパティーベースのフィルターをもとにログをフィルターするロギングソリューションをデプロイできます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

注記

rsyslog パッケージをインストールする必要はありません。RHEL システムロールがデプロイ時に **rsyslog** をインストールするためです。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- name: Deploying files input and configured files output
```

```
hosts: managed-node-01.example.com
roles:
  - rhel-system-roles.logging
vars:
  logging_inputs:
    - name: files_input
      type: basics
  logging_outputs:
    - name: files_output0
      type: files
      property: msg
      property_op: contains
      property_value: error
      path: /var/log/errors.log
    - name: files_output1
      type: files
      property: msg
      property_op: "!contains"
      property_value: error
      path: /var/log/others.log
  logging_flows:
    - name: flow0
      inputs: [files_input]
      outputs: [files_output0, files_output1]
```

この設定を使用すると、**error** 文字列を含むメッセージはすべて **/var/log/errors.log** に記録され、その他のメッセージはすべて **/var/log/others.log** に記録されます。

error プロパティーの値はフィルタリングする文字列に置き換えることができます。

設定に合わせて変数を変更できます。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

1. **/etc/rsyslog.conf** ファイルの構文をテストします。

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

2. システムが **error** 文字列を含むメッセージをログに送信していることを確認します。

   a. テストメッセージを送信します。

```
# logger error
```

b. 以下のように **/var/log/errors.log** ログを表示します。

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

hostname はクライアントシステムのホスト名に置き換えます。ログには、logger コマンドを入力したユーザーのユーザー名 (この場合は **root**) が含まれていることに注意してください。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/logging/** ディレクトリー

13.5. LOGGING システムロールを使用したリモートロギングソリューションの適用

以下の手順に従って、Red Hat Ansible Core Playbook を準備および適用し、リモートロギングソリューションを設定します。この Playbook では、1 つ以上のクライアントが **systemd-journal** からログを取得し、リモートサーバーに転送します。サーバーは、**remote_rsyslog** および **remote_files** からリモート入力を受信し、リモートホスト名によって名付けられたディレクトリーのローカルファイルにログを出力します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

注記

rsyslog パッケージをインストールする必要はありません。RHEL システムロールがデプロイ時に **rsyslog** をインストールするためです。

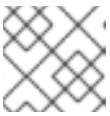
手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- name: Deploying remote input and remote_files output
  hosts: managed-node-01.example.com
  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: managed-node-02.example.com
  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]

[basic_input]
[forward_output0, forward_output1]
```

<host1.example.com> はロギングサーバーに置き換えます。



注記

必要に応じて、Playbook のパラメーターを変更することができます。

警告

ロギングソリューションは、サーバーまたはクライアントシステムの SELinux ポリシーで定義され、ファイアウォールで開放されたポートでしか機能しません。デフォルトの SELinux ポリシーには、ポート 601、514、6514、10514、および 20514 が含まれます。別のポートを使用するには、クライアントシステムおよびサーバーシステムで SELinux ポリシーを変更 します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

1. クライアントとサーバーシステムの両方で、**/etc/rsyslog.conf** ファイルの構文をテストします。

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

2. クライアントシステムがサーバーにメッセージを送信することを確認します。

   a. クライアントシステムで、テストメッセージを送信します。

```
# logger test
```

   b. サーバーシステムで、**/var/log/<host2.example.com>/messages** ログを表示します。次に例を示します。

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

<host2.example.com> は、クライアントシステムのホスト名に置き換えます。ログには、logger コマンドを入力したユーザーのユーザー名 (この場合は **root**) が含まれていることに注意してください。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** ファイル

- **/usr/share/doc/rhel-system-roles/logging/** ディレクトリー

13.6. TLS での LOGGING システムロールの使用

Transport Layer Security (TLS) は、コンピューターネットワーク上でセキュアな通信を可能にするために設計された暗号化プロトコルです。

管理者は、**logging** RHEL システムロールを使用し、Red Hat Ansible Automation Platform を使用したセキュアなログ転送を設定できます。

13.6.1. TLS を使用したクライアントロギングの設定

logging RHEL システムロールを持つ Ansible Playbook を使用して、RHEL クライアントでのロギングを設定し、TLS 暗号化を使用してログをリモートロギングシステムに転送できます。

この手順では、秘密鍵と証明書を作成し、Ansible インベントリーのクライアントグループ内のすべてのホストに TLS を設定します。TLS プロトコルは、メッセージ送信を暗号化し、ネットワーク経由でログを安全に転送します。

注記

証明書を作成するために、Playbook で **certificate** RHEL システムロールを呼び出す必要はありません。**logging** RHEL システムロールがこのロールを自動的に呼び出します。

CA が作成された証明書に署名できるようにするには、管理対象ノードが IdM ドメインに登録されている必要があります。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- 管理対象ノードが IdM ドメインに登録されている。
- 管理対象ノード上で設定するロギングサーバーが RHEL 9.2 以降を実行し、FIPS モードが有効になっている場合、クライアントが Extended Master Secret (EMS) 拡張機能をサポートしているか、TLS 1.3 を使用している。EMS を使用しない TLS 1.2 接続は失敗します。詳細は、ナレッジベースの記事 TLS extension "Extended Master Secret" enforced を参照してください。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Deploying files input and forwards output with certs
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.logging
  vars:
    logging_certificates:
      - name: logging_cert
```

```
      dns: ['localhost', 'www.example.com']
      ca: ipa
  logging_pki_files:
    - ca_cert: /local/path/to/ca_cert.pem
      cert: /local/path/to/logging_cert.pem
      private_key: /local/path/to/logging_cert.pem
  logging_inputs:
    - name: input_name
      type: files
      input_log_path: /var/log/containers/*.log
  logging_outputs:
    - name: output_name
      type: forwards
      target: your_target_host
      tcp_port: 514
      tls: true
      pki_authmode: x509/name
      permitted_server: 'server.example.com'
  logging_flows:
    - name: flow_name
      inputs: [input_name]
      outputs: [output_name]
```

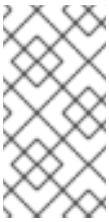
Playbook は以下のパラメーターを使用します。

logging_certificates

このパラメーターの値は、**certificate** RHEL システムロールの **certificate_requests** に渡され、秘密鍵と証明書の作成に使用されます。

logging_pki_files

このパラメーターを使用すると、TLS に使用する CA、証明書、および鍵ファイルを検索するためにロギングで使用するパスとその他の設定 (サブパラメーター **ca_cert**、**ca_cert_src**、**cert**、**cert_src**、**private_key**、**private_key_src**、および **tls** で指定) を設定できます。



注記

logging_certificates を使用してターゲットノードにファイルを作成する場合は、**ca_cert_src**、**cert_src**、および **private_key_src** を使用しないでください。これらは、**logging_certificates** によって作成されていないファイルのコピーに使用されます。

ca_cert

ターゲットノード上の CA 証明書ファイルへのパスを表します。デフォルトのパスは **/etc/pki/tls/certs/ca.pem** で、ファイル名はユーザーが設定します。

cert

ターゲットノード上の証明書ファイルへのパスを表します。デフォルトのパスは **/etc/pki/tls/certs/server-cert.pem** で、ファイル名はユーザーが設定します。

private_key

ターゲットノード上の秘密鍵ファイルへのパスを表します。デフォルトのパスは **/etc/pki/tls/private/server-key.pem** で、ファイル名はユーザーが設定します。

ca_cert_src

ターゲットホストの **ca_cert** で指定された場所にコピーされる、コントロールノード上の CA 証明書ファイルへのパスを表します。**logging_certificates** を使用する場合は、これを使用しないでください。

cert_src

ターゲットホストの **cert** で指定された場所にコピーされる、コントロールノード上の証明書ファイルへのパスを表します。**logging_certificates** を使用する場合は、これを使用しないでください。

private_key_src

ターゲットホストの **private_key** で指定された場所にコピーされる、コントロールノード上の秘密鍵ファイルへのパスを表します。**logging_certificates** を使用する場合は、これを使用しないでください。

tls

このパラメーターを **true** に設定すると、ネットワーク上でログがセキュアに転送されます。セキュアなラッパーが必要ない場合は、**tls: false** に設定します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/logging/** ディレクトリー
- RHEL システムロールを使用した証明書の要求

13.6.2. TLS を使用したサーバーロギングの設定

logging RHEL システムロールを持つ Ansible Playbook を使用して、RHEL サーバーでのロギングを設定し、TLS 暗号化を使用してリモートロギングシステムからログを受信するように設定できます。

この手順では、秘密鍵と証明書を作成し、Ansible インベントリーのサーバーグループ内のすべてのホストに TLS を設定します。

注記

証明書を作成するために、Playbook で **certificate** RHEL システムロールを呼び出す必要はありません。**logging** RHEL システムロールがこのロールを自動的に呼び出します。

CA が作成された証明書に署名できるようにするには、管理対象ノードが IdM ドメインに登録されている必要があります。

前提条件

- コントロールノードと管理対象ノードを準備している。

- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- 管理対象ノードが IdM ドメインに登録されている。
- 管理対象ノード上で設定するロギングサーバーが RHEL 9.2 以降を実行し、FIPS モードが有効になっている場合、クライアントが Extended Master Secret (EMS) 拡張機能をサポートしているか、TLS 1.3 を使用している。EMS を使用しない TLS 1.2 接続は失敗します。詳細は、ナレッジベースの記事 TLS extension "Extended Master Secret" enforced を参照してください。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Deploying remote input and remote_files output with certs
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.logging
  vars:
    logging_certificates:
      - name: logging_cert
        dns: ['localhost', 'www.example.com']
        ca: ipa
    logging_pki_files:
      - ca_cert: /local/path/to/ca_cert.pem
        cert: /local/path/to/logging_cert.pem
        private_key: /local/path/to/logging_cert.pem
    logging_inputs:
      - name: input_name
        type: remote
        tcp_ports: 514
        tls: true
        permitted_clients: ['clients.example.com']
    logging_outputs:
      - name: output_name
        type: remote_files
        remote_log_path: /var/log/remote/%FROMHOST%/%PROGRAMNAME:::secpath-
replace%.log
        async_writing: true
        client_count: 20
        io_buffer_size: 8192
    logging_flows:
      - name: flow_name
        inputs: [input_name]
        outputs: [output_name]
```

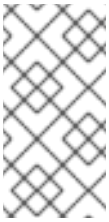
Playbook は以下のパラメーターを使用します。

logging_certificates

このパラメーターの値は、**certificate** RHEL システムロールの **certificate_requests** に渡され、秘密鍵と証明書の作成に使用されます。

logging_pki_files

このパラメーターを使用すると、TLS に使用する CA、証明書、および鍵ファイルを検索するためにロギングで使用するパスとその他の設定 (サブパラメーター **ca_cert**、**ca_cert_src**、**cert**、**cert_src**、**private_key**、**private_key_src**、および **tls** で指定) を設定できます。



注記

logging_certificates を使用してターゲットノードにファイルを作成する場合は、**ca_cert_src**、**cert_src**、および **private_key_src** を使用しないでください。これらは、**logging_certificates** によって作成されていないファイルのコピーに使用されます。

ca_cert

ターゲットノード上の CA 証明書ファイルへのパスを表します。デフォルトのパスは **/etc/pki/tls/certs/ca.pem** で、ファイル名はユーザーが設定します。

cert

ターゲットノード上の証明書ファイルへのパスを表します。デフォルトのパスは **/etc/pki/tls/certs/server-cert.pem** で、ファイル名はユーザーが設定します。

private_key

ターゲットノード上の秘密鍵ファイルへのパスを表します。デフォルトのパスは **/etc/pki/tls/private/server-key.pem** で、ファイル名はユーザーが設定します。

ca_cert_src

ターゲットホストの **ca_cert** で指定された場所にコピーされる、コントロールノード上の CA 証明書ファイルへのパスを表します。**logging_certificates** を使用する場合は、これを使用しないでください。

cert_src

ターゲットホストの **cert** で指定された場所にコピーされる、コントロールノード上の証明書ファイルへのパスを表します。**logging_certificates** を使用する場合は、これを使用しないでください。

private_key_src

ターゲットホストの **private_key** で指定された場所にコピーされる、コントロールノード上の秘密鍵ファイルへのパスを表します。**logging_certificates** を使用する場合は、これを使用しないでください。

tls

このパラメーターを **true** に設定すると、ネットワーク上でログがセキュアに転送されます。セキュアなラッパーが必要ない場合は、**tls: false** に設定します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/logging/** ディレクトリー
- RHEL システムロールを使用した証明書の要求

13.7. RELP での LOGGING システムロールの使用

Reliable Event Logging Protocol (RELP) とは、TCP ネットワークを使用する、データとメッセージロギング用のネットワーキングプロトコルのことです。イベントメッセージを確実に配信するので、メッセージの損失が許されない環境で使用できます。

RELP の送信側はコマンド形式でログエントリーを転送し、受信側は処理後に確認応答します。RELP は、一貫性を保つために、転送されたコマンドごとにトランザクション番号を保存し、各種メッセージの復旧します。

RELP Client と RELP Server の間に、リモートロギングシステムを検討することができます。RELP Client はリモートロギングシステムにログを転送し、RELP Server はリモートロギングシステムから送信されたすべてのログを受け取ります。

管理者は、**logging** RHEL システムロールを使用して、ログエントリーが確実に送受信されるようにロギングシステムを設定できます。

13.7.1. RELP を使用したクライアントロギングの設定

logging RHEL システムロールを使用すると、ローカルマシンにログが記録されている RHEL システムのロギングを設定し、Ansible Playbook を実行して RELP を使用してリモートロギングシステムにログを転送できます。

この手順では、Ansible インベントリーの **client** グループ内の全ホストに RELP を設定します。RELP 設定は Transport Layer Security (TLS) を使用して、メッセージ送信を暗号化し、ネットワーク経由でログを安全に転送します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Deploying basic input and relp output
  hosts: managed-node-01.example.com
  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]
```

Playbook では次の設定を使用します。

target

リモートロギングシステムが稼働しているホスト名を指定する必須パラメーターです。

port

リモートロギングシステムがリッスンしているポート番号です。

tls

ネットワーク上でログをセキュアに転送します。セキュアなラッパーが必要ない場合は、**tls** 変数を **false** に設定します。デフォルトでは **tls** パラメーターは true に設定されますが、RELP を使用する場合には鍵/証明書およびトリプレット {**ca_cert**、**cert**、**private_key**} や {**ca_cert_src**、**cert_src**、**private_key_src**} が必要です。

- {**ca_cert_src**、**cert_src**、**private_key_src**} のトリプレットを設定すると、デフォルトの場所 (**/etc/pki/tls/certs** と **/etc/pki/tls/private**) が、コントロールノードからファイルを転送するため、管理対象ノードの宛先として使用されます。この場合、ファイル名はトリプレットの元の名前と同じです。
- {**ca_cert**、**cert**、**private_key**} トリプレットが設定されている場合は、ファイルはロギング設定の前にデフォルトのパスに配置されている必要があります。
- トリプレットの両方が設定されている場合には、ファイルはコントロールノードのローカルのパスから管理対象ノードの特定のパスへ転送されます。

ca_cert

CA 証明書へのパスを表します。デフォルトのパスは **/etc/pki/tls/certs/ca.pem** で、ファイル名はユーザーが設定します。

cert

証明書へのパスを表します。デフォルトのパスは **/etc/pki/tls/certs/server-cert.pem** で、ファイル名はユーザーが設定します。

private_key

秘密鍵へのパスを表します。デフォルトのパスは **/etc/pki/tls/private/server-key.pem** で、ファイル名はユーザーが設定します。

ca_cert_src

ローカルの CA 証明書ファイルパスを表します。これはターゲットホストにコピーされます。**ca_cert** を指定している場合は、その場所にコピーされます。

cert_src

ローカルの証明書ファイルパスを表します。これはターゲットホストにコピーされます。**cert** を指定している場合には、その証明書が場所にコピーされます。

private_key_src

ローカルキーファイルのパスを表します。これはターゲットホストにコピーされます。**private_key** を指定している場合は、その場所にコピーされます。

pki_authmode

name または **fingerprint** の認証モードを使用できます。

permitted_servers

ロギングクライアントが、TLS 経由での接続およびログ送信を許可するサーバーのリスト。

inputs

ロギング入力ディクショナリーのリスト。

outputs

ロギング出力ディクショナリーのリスト。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/logging/** ディレクトリー

13.7.2. RELP を使用したサーバーログの設定

logging RHEL システムロールを使用して、RHEL システムでのロギングをサーバーとして設定し、Ansible Playbook を実行して RELP を使用してリモートロギングシステムからログを受信できます。

以下の手順では、Ansible インベントリーの **server** グループ内の全ホストに RELP を設定します。RELP 設定は TLS を使用して、メッセージ送信を暗号化し、ネットワーク経由でログを安全に転送します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Deploying remote input and remote_files output
  hosts: managed-node-01.example.com
  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
```

Playbook は以下の設定を使用します。

port

リモートロギングシステムがリッスンしているポート番号です。

tls

ネットワーク上でログをセキュアに転送します。セキュアなラッパーが必要ない場合は、**tls** 変数を **false** に設定します。デフォルトでは **tls** パラメーターは true に設定されますが、RELP を使用する場合には鍵/証明書およびトリプレット {**ca_cert**、**cert**、**private_key**} や {**ca_cert_src**、**cert_src**、**private_key_src**} が必要です。

- {**ca_cert_src**、**cert_src**、**private_key_src**} のトリプレットを設定すると、デフォルトの場所 (**/etc/pki/tls/certs** と **/etc/pki/tls/private**) が、コントロールノードからファイルを転送するため、管理対象ノードの宛先として使用されます。この場合、ファイル名はトリプレットの元の名前と同じです。

- {**ca_cert**、**cert**、**private_key**} トリプレットが設定されている場合は、ファイルはロギング設定の前にデフォルトのパスに配置されている必要があります。

- トリプレットの両方が設定されている場合には、ファイルはコントロールノードのローカルのパスから管理対象ノードの特定のパスへ転送されます。

ca_cert

CA 証明書へのパスを表します。デフォルトのパスは **/etc/pki/tls/certs/ca.pem** で、ファイル名はユーザーが設定します。

cert

証明書へのパスを表します。デフォルトのパスは **/etc/pki/tls/certs/server-cert.pem** で、ファイル名はユーザーが設定します。

private_key

秘密鍵へのパスを表します。デフォルトのパスは **/etc/pki/tls/private/server-key.pem** で、ファイル名はユーザーが設定します。

ca_cert_src

ローカルの CA 証明書ファイルパスを表します。これはターゲットホストにコピーされます。**ca_cert** を指定している場合は、その場所にコピーされます。

cert_src

ローカルの証明書ファイルパスを表します。これはターゲットホストにコピーされます。**cert** を指定している場合には、その証明書が場所にコピーされます。

private_key_src

ローカルキーファイルのパスを表します。これはターゲットホストにコピーされます。**private_key** を指定している場合は、その場所にコピーされます。

pki_authmode

name または **fingerprint** の認証モードを使用できます。

permitted_clients

ロギングサーバーが TLS 経由での接続およびログ送信を許可するクライアントのリスト。

inputs

ロギング入力ディクショナリーのリスト。

outputs

ロギング出力ディクショナリーのリスト。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/logging/** ディレクトリー

第14章 JOURNALD RHEL システムロールを使用した SYSTEMD ジャーナルの設定

journald RHEL システムロールを使用すると、**systemd** ジャーナルを自動化し、Red Hat Ansible Automation Platform を使用して永続的なロギングを設定できます。

14.1. JOURNALD システムロールの変数

journald RHEL システムロールは、**journald** ロギングサービスの動作をカスタマイズするための変数のセットを提供します。ロールには次の変数が含まれます。

journald_persistent

このブール型変数を使用して、ディスクにログファイルを保存する **journald** を **/var/log/journal/** ディレクトリーに設定します。この変数を **true** に設定すると、ログはディスクに保存されます。それ以外の場合は、揮発性メモリーに保存されます。デフォルト値は **false** です。

journald_max_disk_size

この変数を使用して、ジャーナルファイルがディスク上で占有できる最大サイズをメガバイト単位で指定します。**journald.conf(5)** man ページに記載されているデフォルトのサイズ計算を参照してください。

journald_max_files

この変数を使用して、ジャーナルの **journal_max_disk_size** 設定を尊重しながら保持するジャーナルファイルの最大数を指定します。

journald_max_file_size

この変数を使用して、単一ジャーナルファイルの最大サイズをメガバイト単位で指定します。

journald_per_user

このブール型変数を使用して、ログデータをユーザーごとに分けて保持するように **journald** を設定します。デフォルト値は **true** で、特権のないユーザーは、自身のユーザーサービスからシステムログを読み取ることができます。ユーザーごとのジャーナルファイルは、**journald_persistent** 変数が **true** に設定されている場合にのみ、使用できることに注意してください。

journald_compression

このブール型変数を使用して、デフォルトの 512 バイトより大きい **journald** データオブジェクトに圧縮を適用します。デフォルト値は **true** です。

journald_sync_interval

この変数を使用して、**journald** が現在使用されているジャーナルファイルをディスクに同期するまでの時間を分単位で指定します。デフォルトでは、ロールは現在の値を変更しません。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.journald/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/journald/** ディレクトリー
- **journald.conf(5)** man ページ

14.2. JOURNALD システムロールを使用した永続的なロギングの設定

システム管理者は、**journald** RHEL システムロールを使用して永続的なロギングを設定できます。次の例は、以下の目標を達成するために、Playbook で **journald** RHEL システムロール変数を設定する方法を示しています。

- 永続的なロギングの設定
- ジャーナルファイルのディスクスペースの最大サイズの指定
- ログデータをユーザーごとに個別に保持する **journald** の設定
- 同期間隔の定義

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Configure persistent logging
  hosts: managed-node-01.example.com
  vars:
    journald_persistent: true
    journald_max_disk_size: 2048
    journald_per_user: true
    journald_sync_interval: 1
  roles:
    - rhel-system-roles.journald
```

   その結果、**journald** サービスはログをディスク上に最大 2048 MB まで永続的に保存し、ログデータをユーザーごとに分けて保持します。同期は 1 分ごとに行われます。

2. Playbook の構文を検証します。

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

   このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.journald/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/journald/** ディレクトリー

第15章 SSH および SSHD RHEL システムロールを使用したセキュアな通信の設定

管理者は、Ansible Core パッケージを使用すると、**sshd** システムロールを使用して SSH サーバーを設定できます。また、**ssh** システムロールを使用して任意の数の RHEL システムで SSH クライアントを一貫して同時に設定できます。

15.1. SSH サーバーシステムロールの変数

sshd システムロール Playbook では、必要や制限に応じて SSH 設定ファイルのパラメーターを定義できます。

これらの変数が設定されていない場合には、システムロールは RHEL のデフォルト値と同じ **sshd_config** ファイルを作成します。

どのような場合でも、ブール値は **sshd** 設定で適切に **yes** と **no** としてレンダリングされます。リストを使用して複数行の設定項目を定義できます。以下に例を示します。

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

レンダリングは以下のようになります。

```
ListenAddress 0.0.0.0
ListenAddress ::
```

sshd システムロールの変数

sshd_enable

false に設定すると、ロールは完全に無効になります。デフォルトは **true** です。

sshd_skip_defaults

true に設定すると、システムロールがデフォルト値を適用しません。代わりに、**sshd** ディクショナリーまたは **sshd_<OptionName>** 変数のいずれかを使用して、設定のデフォルトの完全なセットを指定します。デフォルトは **false** です。

sshd_manage_service

false に設定すると、サービスは管理されません。つまり、サービスは起動時に有効にならず、開始またはリロードされません。Ansible サービスモジュールは現在、AIX の **enabled** をサポートしていないため、コンテナーまたは AIX 内で実行する場合を除き、デフォルトは **true** になります。

sshd_allow_reload

false に設定すると、設定の変更後に **sshd** はリロードされません。これはトラブルシューティングで役立ちます。変更した設定を適用するには、**sshd** を手動で再読み込みします。AIX を除き、デフォルトは **sshd_manage_service** と同じ値になります。AIX では、**sshd_manage_service** のデフォルトは **false** ですが、**sshd_allow_reload** のデフォルトは **true** です。

sshd_install_service

true に設定すると、ロールは **sshd** サービスのサービスファイルをインストールします。これにより、オペレーティングシステムで提供されるファイルが上書きされます。2 番目のインスタンスを設定し、**sshd_service** 変数も変更する場合を除き、**true** に設定しないでください。デフォルトは **false** です。
ロールは、以下の変数でテンプレートとして参照するファイルを使用します。

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

sshd_service

この変数により **sshd** サービス名が変更されます。これは、2 つ目の **sshd** サービスインスタンスを設定するのに役立ちます。

sshd

設定が含まれるディクショナリー。以下に例を示します。

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

sshd_config(5) は、**sshd** ディクショナリーのすべてのオプションを一覧表示します。

sshd_<OptionName>

ディクショナリーの代わりに、**sshd_** 接頭辞とオプション名で構成される単純な変数を使用して、オプションを定義できます。簡単な変数は、**sshd** ディクショナリーの値をオーバーライドします。以下に例を示します。

```
sshd_Compression: no
```

sshd_config(5) は、**sshd** のすべてのオプションを一覧表示します。

sshd_manage_firewall

デフォルトのポート **22** 以外のポートを使用している場合は、この変数を **true** に設定します。**true** に設定すると、**sshd** ロールは **firewall** ロールを使用してポートアクセスを自動的に管理します。

注記

sshd_manage_firewall 変数はポートのみを追加できます。ポートは削除できません。ポートを削除するには、**firewall** システムロールを直接使用します。**firewall** システムロールを使用してポートを管理する方法の詳細は、システムロールを使用したポートの設定 を参照してください。

sshd_manage_selinux

デフォルトのポート **22** 以外のポートを使用している場合は、この変数を **true** に設定します。**true** に設定すると、**sshd** ロールは **selinux** ロールを使用してポートアクセスを自動的に管理します。

注記

sshd_manage_selinux 変数はポートのみを追加できます。ポートは削除できません。ポートを削除するには、**selinux** システムロールを直接使用します。

sshd_match および sshd_match_1 から sshd_match_9

ディクショナリーのリスト、または Match セクションのディレクトリーのみ。これらの変数は、**sshd** ディクショナリーで定義されている一致ブロックをオーバーライドしないことに注意してください。すべてのソースは作成された設定ファイルに反映されます。

sshd_backup

false に設定すると、元の **sshd_config** ファイルはバックアップされません。デフォルトは **true** です。

sshd システムロールのセカンダリー変数

これらの変数を使用して、サポートされている各プラットフォームに対応するデフォルトを上書きすることができます。

sshd_packages

この変数を使用して、インストール済みパッケージのデフォルトリストを上書きできます。

sshd_config_owner、sshd_config_group、sshd_config_mode

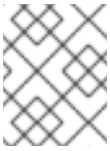
このロールは、これらの変数を使用して生成する **openssh** 設定ファイルの所有権およびパーミッションを設定できます。

sshd_config_file

このロールが作成した **openssh** サーバー設定を保存するパス。

sshd_config_namespace

この変数のデフォルト値は null です。これは、ロールがシステムのデフォルトを含む設定ファイルの内容全体を定義することを意味します。または、この変数を使用して、他のロールから、またはドロップインディレクトリーをサポートしないシステムの 1 つの Playbook 内の複数の場所から、このロールを呼び出すことができます。**sshd_skip_defaults** 変数は無視され、この場合、システムのデフォルトは使用されません。
この変数が設定されている場合、ロールは指定された namespace の下の既存の設定ファイルの設定スニペットに指定する設定を配置します。シナリオにロールを複数回適用する必要がある場合は、アプリケーションごとに異なる namespace を選択する必要があります。



注記

openssh 設定ファイルの制限は引き続き適用されます。たとえば、設定ファイルで指定した最初のオプションだけが、ほとんどの設定オプションで有効です。

技術的には、ロールは他の一致ブロックが含まれていない限り、スニペットを "Match all" ブロックに配置し、既存の設定ファイル内の以前の一致ブロックに関係なく適用されるようにします。これにより、異なるロール呼び出しから競合しないオプションを設定できます。

sshd_binary

openssh の **sshd** 実行可能ファイルへのパス。

sshd_service

sshd サービスの名前。デフォルトでは、この変数には、ターゲットプラットフォームが使用する **sshd** サービスの名前が含まれます。ロールが **sshd_install_service** 変数を使用する場合は、これを使用してカスタムの **sshd** サービスの名前を設定することもできます。

sshd_verify_hostkeys

デフォルトは **auto** です。**auto** に設定すると、生成された設定ファイルに存在するホストキーがすべてリスト表示され、存在しないパスが生成されます。また、パーミッションおよびファイルの所有者はデフォルト値に設定されます。これは、サービスが最初の試行で開始できることを確認するためにデプロイメント段階でロールが使用される場合に便利です。このチェックを無効にするには、この変数を空のリスト **[]** に設定します。

sshd_hostkey_owner, sshd_hostkey_group, sshd_hostkey_mode

これらの変数を使用して、**sshd_verify_hostkeys** からホストキーの所有権とパーミッションを設定します。

sshd_sysconfig

RHEL 8 以前のバージョンをベースとするシステムでは、この変数は **sshd** サービスの追加の詳細を設定します。**true** に設定すると、このロールは **sshd_sysconfig_override_crypto_policy** および **sshd_sysconfig_use_strong_rng** 変数に基づいて、**/etc/sysconfig/sshd** 設定ファイルも管理します。デフォルトは **false** です。

sshd_sysconfig_override_crypto_policy

RHEL 8 では、これを **true** に設定すると、**sshd** ディクショナリーまたは **sshd_<OptionName>** 形式で次の設定オプションを使用して、システム全体の暗号化ポリシーをオーバーライドできます。

- **Ciphers**
- **MACs**
- **GSSAPIKexAlgorithms**
- **GSSAPIKeyExchange** (FIPS-only)
- **KexAlgorithms**
- **HostKeyAlgorithms**
- **PubkeyAcceptedKeyTypes**
- **CASignatureAlgorithms**
  デフォルトは **false** です。

  RHEL 9 では、この変数は影響を及ぼしません。代わりに、**sshd** ディクショナリーまたは **sshd_<OptionName>** 形式で次の設定オプションを使用して、システム全体の暗号化ポリシーをオーバーライドできます。

- **Ciphers**
- **MACs**
- **GSSAPIKexAlgorithms**
- **GSSAPIKeyExchange** (FIPS-only)
- **KexAlgorithms**
- **HostKeyAlgorithms**
- **PubkeyAcceptedAlgorithms**
- **HostbasedAcceptedAlgorithms**
- **CASignatureAlgorithms**
- **RequiredRSASize**
  sshd_config_file 変数で定義されたドロップインディレクトリーのカスタム設定ファイルに、これらのオプションを入力する場合は、暗号化ポリシーを含む **/etc/ssh/sshd_config.d/50-redhat.conf** ファイルより辞書順に前にあるファイル名を使用します。

sshd_sysconfig_use_strong_rng

RHEL 8 以前のバージョンをベースとするシステムでは、この変数により、**sshd** が、引数として指定されたバイト数を使用して **openssl** 乱数ジェネレーターを強制的に再シードできます。デフォルトは **0** で、この機能を無効にします。システムにハードウェア乱数ジェネレーターがない場合は、この機能を有効にしないでください。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.sshd/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/sshd/** ディレクトリー

15.2. SSHD システムロールを使用した OPENSSH サーバーの設定

sshd システムロールを使用して、Ansible Playbook を実行し、複数の SSH サーバーを設定できます。

注記

sshd システムロールは、SSH および SSHD 設定を変更する他のシステムロール (ID 管理 RHEL システムロールなど) とともに使用できます。設定が上書きされないようにするには、**sshd** ロールがネームスペース (RHEL 8 以前のバージョン) またはドロップインディレクトリー (RHEL 9) を使用していることを確認してください。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: SSH server configuration
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure sshd to prevent root and password login except from particular subnet
      ansible.builtin.include_role:
        name: rhel-system-roles.sshd
      vars:
        sshd:
          PermitRootLogin: no
          PasswordAuthentication: no
          Match:
            - Condition: "Address 192.0.2.0/24"
              PermitRootLogin: yes
              PasswordAuthentication: yes
```

Playbook は、以下のように、マネージドノードを SSH サーバーとして設定します。

- パスワードと **root** ユーザーのログインが無効である
- **192.0.2.0/24** のサブネットからのパスワードおよび **root** ユーザーのログインのみが有効である

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

1. SSH サーバーにログインします。

```
$ ssh <username>@<ssh_server>
```

2. SSH サーバー上の **sshd_config** ファイルの内容を確認します。

```
$ cat /etc/ssh/sshd_config.d/00-ansible_system_role.conf
#
# Ansible managed
#
PasswordAuthentication no
PermitRootLogin no
Match Address 192.0.2.0/24
  PasswordAuthentication yes
  PermitRootLogin yes
```

3. **192.0.2.0/24** サブネットから root としてサーバーに接続できることを確認します。

a. IP アドレスを確認します。

```
$ hostname -I
192.0.2.1
```

IP アドレスが **192.0.2.1** - **192.0.2.254** 範囲にある場合は、サーバーに接続できます。

b. **root** でサーバーに接続します。

```
$ ssh root@<ssh_server>
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.sshd/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/sshd/** ディレクトリー

15.3. 非排他的設定に SSHD システムロールを使用する

通常、**sshd** システムロールを適用すると、設定全体が上書きされます。これは、たとえば別のシステムロールや Playbook などを使用して、以前に設定を調整している場合に問題を生じる可能性があります。他のオプションを維持しながら、選択した設定オプションにのみ **sshd** システムロールを適用するには、非排他的設定を使用できます。

非排他的設定は、以下を使用して適用できます。

- RHEL 8 以前では、設定スニペットを使用します。
- RHEL 9 以降では、ドロップインディレクトリー内のファイルを使用します。デフォルトの設定ファイルは、**/etc/ssh/sshd_config.d/00-ansible_system_role.conf** としてドロップインディレクトリーにすでに配置されています。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。
   - RHEL 8 以前を実行する管理対象ノードの場合:

```
---
- name: Non-exclusive sshd configuration
  hosts: managed-node-01.example.com
  tasks:
    - name: <Configure SSHD to accept some useful environment variables>
      ansible.builtin.include_role:
        name: rhel-system-roles.sshd
      vars:
        sshd_config_namespace: <my-application>
        sshd:
          # Environment variables to accept
          AcceptEnv:
            LANG
            LS_COLORS
            EDITOR
```

   - RHEL 9 以降を実行する管理対象ノードの場合:

```
- name: Non-exclusive sshd configuration
  hosts: managed-node-01.example.com
  tasks:
    - name: <Configure sshd to accept some useful environment variables>
      ansible.builtin.include_role:
        name: rhel-system-roles.sshd
      vars:
```

```
    sshd_config_file: /etc/ssh/sshd_config.d/<42-my-application>.conf
    sshd:
      # Environment variables to accept
      AcceptEnv:
        LANG
        LS_COLORS
        EDITOR
```

sshd_config_file 変数では、**sshd** システムロールによる設定オプションの書き込み先の **.conf** ファイルを定義します。設定ファイルが適用される順序を指定するには、2 桁の接頭辞 (例: **42-**) を使用します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

- SSH サーバーの設定を確認します。
  - RHEL 8 以前を実行する管理対象ノードの場合:

```
# cat /etc/ssh/sshd_config.d/42-my-application.conf
# Ansible managed
#
AcceptEnv LANG LS_COLORS EDITOR
```

  - RHEL 9 以降を実行する管理対象ノードの場合:

```
# cat /etc/ssh/sshd_config
...
# BEGIN sshd system role managed block: namespace <my-application>
Match all
  AcceptEnv LANG LS_COLORS EDITOR
# END sshd system role managed block: namespace <my-application>
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.sshd/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/sshd/** ディレクトリー

15.4. システムロールを使用して SSH サーバー上のシステム全体の暗号化ポリシーをオーバーライドする

sshd RHEL システムロールを使用して、SSH サーバー上のシステム全体の暗号化ポリシーをオーバーライドできます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
- name: Overriding the system-wide cryptographic policy
  hosts: managed-node-01.example.com
  roles:
    - rhel_system_roles.sshd
  vars:
    sshd_sysconfig: true
    sshd_sysconfig_override_crypto_policy: true
    sshd_KexAlgorithms: ecdh-sha2-nistp521
    sshd_Ciphers: aes256-ctr
    sshd_MACs: hmac-sha2-512-etm@openssh.com
    sshd_HostKeyAlgorithms: rsa-sha2-512,rsa-sha2-256
```

   - **sshd_KexAlgorithms**: **ecdh-sha2-nistp256**、**ecdh-sha2-nistp384**、**ecdh-sha2-nistp521**、**diffie-hellman-group14-sha1**、**diffie-hellman-group-exchange-sha256** などの鍵交換アルゴリズムを選択できます。
   - **sshd_Ciphers**: **aes128-ctr**、**aes192-ctr**、**aes256-ctr** などの暗号を選択できます。
   - **sshd_MACs**: **hmac-sha2-256**、**hmac-sha2-512**、**hmac-sha1** などの MAC を選択できます。
   - **sshd_HostKeyAlgorithms**: **ecdsa-sha2-nistp256**、**ecdsa-sha2-nistp384**、**ecdsa-sha2-nistp521**、**ssh-rsa**、**ssh-dss** などの公開鍵アルゴリズムを選択できます。

   RHEL 9 管理対象ノードでは、システムロールによって設定が **/etc/ssh/sshd_config.d/00-ansible_system_role.conf** ファイルに書き込まれ、暗号化オプションが自動的に適用されます。**sshd_config_file** 変数を使用してファイルを変更できます。ただし、設定を確実に有効にするために、設定された暗号化ポリシーが含まれる **/etc/ssh/sshd_config.d/50-redhat.conf** ファイルよりも辞書式順序で前に来るようなファイル名を使用してください。

   RHEL 8 管理対象ノードでは、**sshd_sysconfig_override_crypto_policy** 変数と **sshd_sysconfig** 変数を **true** に設定してオーバーライドを有効にする必要があります。

2. Playbook の構文を検証します。

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

   このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

- 詳細なログを表示する SSH 接続を使用して手順が成功したかどうかを検証し、定義した変数を以下の出力で確認できます。

```
$ ssh -vvv <ssh_server>
...
debug2: peer server KEXINIT proposal
debug2: KEX algorithms: ecdh-sha2-nistp521
debug2: host key algorithms: rsa-sha2-512,rsa-sha2-256
debug2: ciphers ctos: aes256-ctr
debug2: ciphers stoc: aes256-ctr
debug2: MACs ctos: hmac-sha2-512-etm@openssh.com
debug2: MACs stoc: hmac-sha2-512-etm@openssh.com
...
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.sshd/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/sshd/** ディレクトリー

15.5. SSH システムロールの変数

ssh システムロール Playbook では、必要や制限に応じてクライアント SSH 設定ファイルのパラメーターを定義できます。

これらの変数が設定されていない場合には、システムロールは RHEL のデフォルト値と同じグローバル **ssh_config** ファイルを作成します。

どのような場合でも、ブール値は **ssh** 設定で適切に **yes** または **no** とレンダリングされます。リストを使用して複数行の設定項目を定義できます。以下に例を示します。

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

レンダリングは以下のようになります。

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

注記

設定オプションでは、大文字と小文字が区別されます。

ssh システムロールの変数

ssh_user

システムロールでユーザー固有の設定を変更するように、既存のユーザー名を定義できます。ユーザー固有の設定は、指定したユーザーの **~/.ssh/config** に保存されます。デフォルト値は null で、すべてのユーザーに対するグローバル設定を変更します。

ssh_skip_defaults

デフォルトは **auto** です。**auto** に設定すると、システムロールはシステム全体の設定ファイル **/etc/ssh/ssh_config** を読み取り、そこで定義した RHEL のデフォルトを保持します。**ssh_drop_in_name** 変数を定義してドロップイン設定ファイルを作成すると、**ssh_skip_defaults** 変数が自動的に無効化されます。

ssh_drop_in_name

システム全体のドロップインディレクトリーに置かれたドロップイン設定ファイルの名前を定義します。この名前は、変更する設定ファイルを参照するテンプレート **/etc/ssh/ssh_config.d/{ssh_drop_in_name}.conf** で使用されます。システムがドロップインディレクトリーに対応していない場合、デフォルト値は null です。システムがドロップインディレクトリーに対応している場合、デフォルト値は **00-ansible** です。

警告

システムがドロップインディレクトリーに対応していない場合は、このオプションを設定すると、プレイに失敗します。

推奨される形式は **NN-name** です。**NN** は、設定ファイルの指定に使用する 2 桁の番号で、 **name** はコンテンツまたはファイルの所有者を示す名前になります。

ssh

設定オプションとその値が含まれる dict。

ssh_OptionName

dict の代わりに、**ssh_** 接頭辞とオプション名で設定される単純な変数を使用してオプションを定義できます。簡単な変数は、**ssh** dict の値を上書きします。

ssh_additional_packages

このロールは、一般的なユースケースに必要な **openssh** パッケージおよび **openssh-clients** パッケージを自動的にインストールします。ホストベースの認証用に **openssh-keysign** などの追加のパッケージをインストールする必要がある場合は、この変数で指定できます。

ssh_config_file

ロールが生成した設定ファイルを保存するパス。デフォルト値:

- システムにドロップインディレクトリーがある場合、デフォルト値は **/etc/ssh/ssh_config.d/{ssh_drop_in_name}.conf** テンプレートで定義されます。
- システムにドロップインディレクトリーがない場合、デフォルト値は **/etc/ssh/ssh_config** になります。
- **ssh_user** 変数が定義されている場合、デフォルト値は **~/.ssh/config** になります。

ssh_config_owner, ssh_config_group, ssh_config_mode

作成した設定ファイルの所有者、グループ、およびモード。デフォルトでは、ファイルの所有者は **root:root** で、モードは **0644** です。**ssh_user** が定義されている場合、モードは **0600** で、owner と group は **ssh_user** 変数で指定したユーザー名から派生します。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.ssh/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/ssh/** ディレクトリー

15.6. SSH システムロールを使用した OPENSSH クライアントの設定

ssh システムロールを使用して、Ansible Playbook を実行し、複数の SSH クライアントを設定できます。

注記

ssh システムロールは、SSH および SSHD 設定を変更する他のシステムロール (ID 管理 RHEL システムロールなど) とともに使用できます。設定が上書きされないようにするには、**ssh** ロールがドロップインディレクトリーを使用していること (RHEL 8 以降ではデフォルト) を確認してください。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: SSH client configuration
  hosts: managed-node-01.example.com
  tasks:
    - name: "Configure ssh clients"
      ansible.builtin.include_role:
        name: rhel-system-roles.ssh
      vars:
        ssh_user: root
        ssh:
          Compression: true
          GSSAPIAuthentication: no
          ControlMaster: auto
          ControlPath: ~/.ssh/.cm%C
          Host:
            - Condition: example
              Hostname: server.example.com
              User: user1
        ssh_ForwardX11: no
```

この Playbook は、以下の設定を使用して、マネージドノードで **root** ユーザーの SSH クライアント設定を行います。

- 圧縮が有効になっている。
- ControlMaster multiplexing が **auto** に設定されている。
- **server.example.com** ホストに接続するための **example** エイリアスが **user1** である。
- **example** ホストエイリアスが作成済みで、このエイリアスがユーザー名 **user1** を持つ **server.example.com** ホストへの接続を表している。
- X11 転送が無効化されている。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

- SSH 設定ファイルを表示して、管理対象ノードの設定が正しいことを確認します。

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

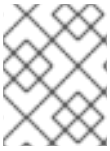
関連情報

- **/usr/share/ansible/roles/rhel-system-roles.ssh/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/ssh/** ディレクトリー

第16章 VPN RHEL システムロールを使用した IPSEC による VPN 接続の設定

vpn システムロールを使用すると、Red Hat Ansible Automation Platform を使用して RHEL システムで VPN 接続を設定できます。これを使用して、ホスト間、ネットワーク間、VPN リモートアクセスサーバー、およびメッシュ設定をセットアップできます。

ホスト間接続の場合、ロールは、必要に応じてキーを生成するなど、デフォルトのパラメーターを使用して、**vpn_connections** のリスト内のホストの各ペア間に VPN トンネルを設定します。または、リストされているすべてのホスト間にオポチュニスティックメッシュ設定を作成するように設定することもできます。このロールは、**hosts** の下にあるホストの名前が Ansible インベントリーで使用されているホストの名前と同じであり、それらの名前を使用してトンネルを設定できることを前提としています。



注記

vpn RHEL システムロールは、現在 VPN プロバイダーとして、IPsec 実装である Libreswan のみをサポートしています。

16.1. VPN システムロールを使用して IPSEC によるホスト間 VPN を作成する

vpn システムロールを使用して、コントロールノードで Ansible Playbook を実行することにより、ホスト間接続を設定できます。これにより、インベントリーファイルにリストされているすべての管理対象ノードが設定されます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
- name: Host to host VPN
  hosts: managed-node-01.example.com, managed-node-02.example.com
  roles:
    - rhel-system-roles.vpn
  vars:
    vpn_connections:
      - hosts:
          managed-node-01.example.com:
          managed-node-02.example.com:
    vpn_manage_firewall: true
    vpn_manage_selinux: true
```

この Playbook は、システムロールによって自動生成されたキーを使用した事前共有キー認証を使用して、接続 **managed-node-01.example.com-to-managed-node-02.example.com** を設定します。**vpn_manage_firewall** と **vpn_manage_selinux** は両方とも **true** に設定されているた

め、**vpn** ロールは **firewall** ロールと **selinux** ロールを使用して、**vpn** ロールが使用するポートを管理します。

管理対象ホストから、インベントリーファイルにリストされていない外部ホストへの接続を設定するには、ホストの **vpn_connections** リストに次のセクションを追加します。

```
vpn_connections:
  - hosts:
      managed-node-01.example.com:
      <external_node>:
        hostname: <IP_address_or_hostname>
```

これにより、追加の接続 **managed-node-01.example.com-to-<external_node>** が 1 つ設定されます。

注記

接続は管理対象ノードでのみ設定され、外部ノードでは設定されません。

2. 必要に応じて、**vpn_connections** 内の追加セクション (コントロールプレーンやデータプレーンなど) を使用して、管理対象ノードに複数の VPN 接続を指定できます。

```
- name: Multiple VPN
  hosts: managed-node-01.example.com, managed-node-02.example.com
  roles:
    - rhel-system-roles.vpn
  vars:
    vpn_connections:
      - name: control_plane_vpn
        hosts:
          managed-node-01.example.com:
            hostname: 192.0.2.0 # IP for the control plane
          managed-node-02.example.com:
            hostname: 192.0.2.1
      - name: data_plane_vpn
        hosts:
          managed-node-01.example.com:
            hostname: 10.0.0.1 # IP for the data plane
          managed-node-02.example.com:
            hostname: 10.0.0.2
```

3. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

4. Playbook を実行します。

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

検証

1. マネージドノードで、接続が正常にロードされていることを確認します。

```
# ipsec status | grep <connection_name>
```

<connection_name>を、このノードからの接続の名前 (たとえば、**managed_node1-to-managed_node2**) に置き換えます。

注記

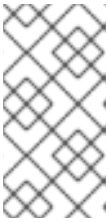
デフォルトでは、ロールは、各システムの観点から作成する接続ごとにわかりやすい名前を生成します。たとえば、**managed_node1** と **managed_node2** との間の接続を作成するときに、**managed_node1** 上のこの接続のわかりやすい名前は **managed_node1-to-managed_node2** ですが、**managed_node2** では、この接続の名前は **managed_node2-to-managed_node1** となります。

2. マネージドノードで、接続が正常に開始されたことを確認します。

```
# ipsec trafficstatus | grep <connection_name>
```

3. オプション: 接続が正常にロードされない場合は、次のコマンドを入力して手動で接続を追加します。これにより、接続の確立に失敗した理由を示す、より具体的な情報が提供されます。

```
# ipsec auto --add <connection_name>
```



注記

接続のロードおよび開始のプロセスで発生する可能性のあるエラーは、**/var/log/pluto.log** ファイルに報告されます。これらのログは解析が難しいため、代わりに接続を手動で追加して、標準出力からログメッセージを取得してください。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.vpn/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/vpn/** ディレクトリー

16.2. VPN システムロールを使用して IPSEC によるオポチュニスティックメッシュ VPN 接続を作成する

vpn システムロールを使用して、コントロールノードで Ansible Playbook を実行することにより、認証に証明書を使用するオポチュニスティックメッシュ VPN 接続を設定できます。これにより、インベントリーファイルにリストされているすべての管理対象ノードが設定されます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

/etc/ipsec.d/ ディレクトリーの IPsec ネットワークセキュリティーサービス (NSS) 暗号ライブラリーに、必要な証明書が含まれている。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
- name: Mesh VPN
  hosts: managed-node-01.example.com, managed-node-02.example.com, managed-node-03.example.com
  roles:
    - rhel-system-roles.vpn
  vars:
    vpn_connections:
      - opportunistic: true
        auth_method: cert
        policies:
          - policy: private
            cidr: default
          - policy: private-or-clear
            cidr: 198.51.100.0/24
          - policy: private
            cidr: 192.0.2.0/24
          - policy: clear
            cidr: 192.0.2.7/32
    vpn_manage_firewall: true
    vpn_manage_selinux: true
```

証明書による認証は、Playbook で **auth_method: cert** パラメーターを定義することによって設定されます。デフォルトでは、ノード名が証明書のニックネームとして使用されます。この例では、**managed-node-01.example.com** です。インベントリーで **cert_name** 属性を使用して、さまざまな証明書名を定義できます。

この例の手順では、Ansible Playbook の実行元のシステムであるコントロールノードが、両方の管理対象ノードと同じ Classless Inter-Domain Routing (CIDR) 番号 (192.0.2.0/24) を共有し、IP アドレス 192.0.2.7 を持ちます。したがって、コントロールノードは、CIDR 192.0.2.0/24 用に自動的に作成されるプライベートポリシーに該当します。

再生中の SSH 接続の損失を防ぐために、コントロールノードの明確なポリシーがポリシーのリストに含まれています。ポリシーリストには、CIDR がデフォルトと等しい項目もあることに注意してください。これは、この Playbook がデフォルトポリシーのルールを上書きして、private-or-clear ではなく private にするためです。

vpn_manage_firewall と **vpn_manage_selinux** は両方とも **true** に設定されているため、**vpn** ロールは **firewall** ロールと **selinux** ロールを使用して、**vpn** ロールが使用するポートを管理します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.vpn/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/vpn/** ディレクトリー

第17章 CRYPTO-POLICIES RHEL システムロールを使用したカスタム暗号化ポリシーの設定

管理者は、**crypto_policies** RHEL システムロールを使用して、Ansible Core パッケージを使用し、さまざまなシステムのカスタム暗号化ポリシーを迅速かつ一貫して設定できます。

17.1. CRYPTO_POLICIES システムロールの変数とファクト

crypto_policies システムロール Playbook では、必要や制限に応じて **crypto_policies** 設定ファイルのパラメーターを定義できます。

変数を設定しない場合には、システムロールではシステムが設定されず、ファクトのみが報告されます。

crypto_policies システムロールの一部の変数

crypto_policies_policy

マネージドノードにシステムロールを適用する暗号化ポリシーを決定します。異なる暗号化ポリシーの詳細は、システム全体の暗号化ポリシー を参照してください。

crypto_policies_reload

yes に設定すると、暗号化ポリシーの適用後に、影響を受けるサービス (現在 **ipsec**、**バインド**、および **sshd** サービス) でリロードされます。デフォルトは **yes** です。

crypto_policies_reboot_ok

yes に設定されており、システムロールで暗号化ポリシーを変更した後に再起動が必要な場合には、**crypto_policies_reboot_required** を **yes** に設定します。デフォルトは **no** です。

crypto_policies システムロールにより設定されるファクト

crypto_policies_active

現在選択されているポリシーをリスト表示します。

crypto_policies_available_policies

システムで利用可能なすべてのポリシーを表示します。

crypto_policies_available_subpolicies

システムで利用可能なすべてのサブポリシーを表示します。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.crypto_policies/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/crypto_policies/** ディレクトリー
- システム全体のカスタム暗号化ポリシーの作成および設定

17.2. CRYPTO_POLICIES システムロールを使用したカスタム暗号化ポリシーの設定

crypto_policies システムロールを使用して、単一のコントロールノードから多数の管理対象ノードを一貫して設定できます。

前提条件

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Configure crypto policies
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure crypto policies
      ansible.builtin.include_role:
        name: rhel-system-roles.crypto_policies
      vars:
        - crypto_policies_policy: FUTURE
        - crypto_policies_reboot_ok: true
```

FUTURE の値は、任意の暗号化ポリシー (例: **DEFAULT**、**LEGACY**、および **FIPS:OSPP**) に置き換えることができます。

crypto_policies_reboot_ok: true を設定すると、システムロールが暗号化ポリシーを変更した後にシステムが再起動します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

1. コントロールノードで、たとえば **verify_playbook.yml** という名前の別の Playbook を作成します。

```
---
- name: Verification
  hosts: managed-node-01.example.com
  tasks:
    - name: Verify active crypto policy
      ansible.builtin.include_role:
        name: rhel-system-roles.crypto_policies
    - debug:
        var: crypto_policies_active
```

2. Playbook の構文を検証します。

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

3. Playbook を実行します。

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

crypto_policies_active 変数は、管理対象ノードでアクティブなポリシーを示します。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.crypto_policies/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/crypto_policies/** ディレクトリー

第18章 RHEL システムロールを使用した NBDE の設定

18.1. NBDE_CLIENT および NBDE_SERVER システムロールの概要 (CLEVIS および TANG)

RHEL システムロールは、複数の RHEL システムをリモートで管理するための一貫した設定インターフェイスを提供する Ansible ロールおよびモジュールのコレクションです。

Clevis および Tang を使用した PBD (Policy-Based Decryption) ソリューションの自動デプロイメント用 Ansible ロールを使用することができます。**rhel-system-roles** パッケージには、これらのシステムロール、関連する例、リファレンスドキュメントが含まれます。

nbde_client システムロールにより、複数の Clevis クライアントを自動的にデプロイできます。**nbde_client** ロールは、Tang バインディングのみをサポートしており、現時点では TPM2 バインディングには使用できない点に留意してください。

nbde_client ロールには、LUKS を使用して暗号化済みのボリュームが必要です。このロールは、LUKS 暗号化ボリュームの 1 つ以上の Network-Bound (NBDE) サーバー (Tang サーバー) へのバインドに対応します。パスフレーズを使用して既存のボリュームの暗号化を保持するか、削除できます。パスフレーズを削除したら、NBDE だけを使用してボリュームのロックを解除できます。これは、システムのプロビジョニング後に削除する必要がある一時鍵またはパスワードを使用して、ボリュームが最初に暗号化されている場合に役立ちます。

パスフレーズと鍵ファイルの両方を指定する場合には、ロールは最初に指定した内容を使用します。有効なバインディングが見つからない場合は、既存のバインディングからパスフレーズの取得を試みます。

PBD では、デバイスをスロットにマッピングするものとしてバインディングを定義します。つまり、同じデバイスに複数のバインディングを指定できます。デフォルトのスロットは 1 です。

nbde_client ロールでは、**state** 変数も指定できます。新しいバインディングを作成するか、既存のバインディングを更新する場合は、**present** を使用します。**clevis luks bind** とは異なり、**state: present** を使用してデバイススロットにある既存のバインディングを上書きすることもできます。**absent** に設定すると、指定したバインディングが削除されます。

nbde_client システムロールを使用すると、自動ディスク暗号化ソリューションの一部として、Tang サーバーをデプロイして管理できます。このロールは以下の機能をサポートします。

- Tang 鍵のローテーション
- Tang 鍵のデプロイおよびバックアップ

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.nbde_server/README.md** ファイル
- **/usr/share/ansible/roles/rhel-system-roles.nbde_client/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/nbde_server/** ディレクトリー
- **/usr/share/doc/rhel-system-roles/nbde_client/** ディレクトリー

18.2. 複数の TANG サーバーのセットアップに NBDE_SERVER システムロールを使用する

以下の手順に従って、Tang サーバー設定を含む Ansible Playbook を準備および適用します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.nbde_server
  vars:
    nbde_server_rotate_keys: yes
    nbde_server_manage_firewall: true
    nbde_server_manage_selinux: true
```

このサンプル Playbook により、Tang サーバーのデプロイと鍵のローテーションが実行されます。

nbde_server_manage_firewall と **nbde_server_manage_selinux** が両方とも **true** に設定されている場合、**nbde_server** ロールは **firewall** ロールと **selinux** ロールを使用して、**nbde_server** ロールが使用するポートを管理します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

- Clevis がインストールされているシステムで **grubby** ツールを使用して、システム起動時の早い段階で Tang ピンのネットワークを利用できるようにするには、次のコマンドを実行します。

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

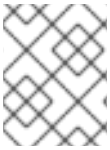
関連情報

- **/usr/share/ansible/roles/rhel-system-roles.nbde_server/README.md** ファイル

- **/usr/share/doc/rhel-system-roles/nbde_server/** ディレクトリー

18.3. NBDE_CLIENT RHEL システムロールを使用した複数の CLEVIS クライアントのセットアップ

nbde_client RHEL システムロールを使用すると、Clevis クライアント設定を含む Ansible Playbook を複数のシステムで準備および適用できます。



注記

nbde_client システムロールは、Tang バインディングのみをサポートします。したがって、TPM2 バインディングには使用できません。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.nbde_client
  vars:
    nbde_client_bindings:
      - device: /dev/rhel/root
        encryption_key_src: /etc/luks/keyfile
        servers:
          - http://server1.example.com
          - http://server2.example.com
      - device: /dev/rhel/swap
        encryption_key_src: /etc/luks/keyfile
        servers:
          - http://server1.example.com
          - http://server2.example.com
```

このサンプル Playbook は、2 台の Tang サーバーのうち少なくとも 1 台が利用可能な場合に、LUKS で暗号化した 2 つのボリュームを自動的にアンロックするように Clevis クライアントを設定します。

nbde_client システムロールは、動的ホスト設定プロトコル (DHCP) を使用する場合のみをサポートします。静的 IP 設定のクライアントに NBDE を使用するには、次の Playbook を使用します。

```
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.nbde_client
  vars:
```

```
    nbde_client_bindings:
      - device: /dev/rhel/root
        encryption_key_src: /etc/luks/keyfile
        servers:
          - http://server1.example.com
          - http://server2.example.com
      - device: /dev/rhel/swap
        encryption_key_src: /etc/luks/keyfile
        servers:
          - http://server1.example.com
          - http://server2.example.com
  tasks:
    - name: Configure a client with a static IP address during early boot
      ansible.builtin.command:
        cmd: grubby --update-kernel=ALL --args='GRUB_CMDLINE_LINUX_DEFAULT="ip={{
<ansible_default_ipv4.address> }}::{{ <ansible_default_ipv4.gateway> }}:{{
<ansible_default_ipv4.netmask> }}::{{ <ansible_default_ipv4.alias> }}:none"'
```

この Playbook の **<ansible_default_ipv4.*>** 文字列は、ネットワークの IP アドレス (**ip={{ 192.0.2.10 }}::{{ 192.0.2.1 }}:{{ 255.255.255.0 }}::{{ ens3 }}:none**) に置き換えます。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.nbde_client/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/nbde_client/** ディレクトリー
- Looking forward to Linux network configuration in the initial ramdisk (initrd) article

第19章 RHEL システムロールを使用した証明書の要求

certificate システムロールを使用すると、証明書を発行および管理できます。

19.1. CERTIFICATE システムロール

certificate システムロールを使用すると、Ansible Core を使用して TLS および SSL 証明書の発行と更新を管理できます。

ロールは **certmonger** を証明書プロバイダーとして使用し、自己署名証明書の発行と更新、および IdM 統合認証局 (CA) の使用を現時点でサポートしています。

certificate システムロールを含む Ansible Playbook では、次の変数を使用できます。

certificate_wait
タスクが証明書を発行するまで待機するかどうかを指定します。

certificate_requests
発行する各証明書とそのパラメーターを表すには、次のコマンドを実行します。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.certificate/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/certificate/** ディレクトリー

19.2. CERTIFICATE システムロールを使用した新しい自己署名証明書の要求

certificate システムロールを使用すると、Ansible Core を使用して自己署名証明書を発行できます。

このプロセスは、**certmonger** プロバイダーを使用し、**getcert** コマンドで証明書を要求します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.certificate
  vars:
    certificate_requests:
      - name: mycert
        dns: "*.example.com"
        ca: self-sign
```

- **name** パラメーターを希望する証明書の名前 (**mycert** など) に設定します。
- **dns** パラメーターを ***.example.com** などの証明書に含むドメインに設定します。
- **ca** パラメーターを **self-sign** に設定します。

デフォルトでは、**certmonger** は有効期限が切れる前に証明書の更新を自動的に試行します。これは、Ansible Playbook の **auto_renew** パラメーターを **no** に設定すると無効にできます。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.certificate/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/certificate/** ディレクトリー

19.3. CERTIFICATE システムロールを使用した IDM CA からの新しい証明書の要求

certificate システムロールを使用すると、統合認証局 (CA) を持つ IdM サーバーを使用しながら、**ansible-core** を使用して証明書を発行できます。したがって、IdM を CA として使用する場合に、複数のシステムの証明書トラストチェーンを効率的かつ一貫して管理できます。

このプロセスは、**certmonger** プロバイダーを使用し、**getcert** コマンドで証明書を要求します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.certificate
  vars:
    certificate_requests:
```

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

- **name** パラメーターを希望する証明書の名前 (**mycert** など) に設定します。
- **dns** パラメーターをドメインに設定し、証明書に追加します (例: **www.example.com**)。
- **principal** パラメーターを設定し、Kerberos プリンシパルを指定します (例: **HTTP/www.example.com@EXAMPLE.COM**)。
- **ca** パラメーターを **ipa** に設定します。

デフォルトでは、**certmonger** は有効期限が切れる前に証明書の更新を自動的に試行します。これは、Ansible Playbook の **auto_renew** パラメーターを **no** に設定すると無効にできます。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.certificate/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/certificate/** ディレクトリー

19.4. CERTIFICATE システムロールを使用して証明書発行前または発行後に実行するコマンドを指定する

certificate ロールでは、Ansible Core を使用して、証明書の発行または更新の前後にコマンドを実行できます。

以下の例では、管理者が **www.example.com** の自己署名証明書を発行または更新する前に **httpd** サービスを停止し、後で再起動します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.certificate
  vars:
    certificate_requests:
      - name: mycert
        dns: www.example.com
        ca: self-sign
        run_before: systemctl stop httpd.service
        run_after: systemctl start httpd.service
```

   - **name** パラメーターを希望する証明書の名前 (**mycert** など) に設定します。
   - **dns** パラメーターをドメインに設定し、証明書に追加します (例: **www.example.com**)。
   - **ca** パラメーターを証明書を発行する際に使用する CA に設定します (例: **self-sign**)。
   - この証明書を発行または更新する前に、**run_before** パラメーターを実行するコマンドに設定します (例: **systemctl stop httpd.service**)。
   - この証明書を発行または更新した後に、**run_after** パラメーターを実行するコマンドに設定します (例: **systemctl start httpd.service**)。

   デフォルトでは、**certmonger** は有効期限が切れる前に証明書の更新を自動的に試行します。これは、Ansible Playbook の **auto_renew** パラメーターを **no** に設定すると無効にできます。

2. Playbook の構文を検証します。

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

   このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.certificate/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/certificate/** ディレクトリー

第20章 KDUMP RHEL システムロールを使用した自動クラッシュダンプの設定

Ansible を使用して kdump を管理するには、RHEL 9 で使用可能な RHEL システムロールの 1 つである **kdump** ロールを使用できます。

kdump ロールを使用すると、後で分析するためにシステムのメモリーの内容を保存する場所を指定できます。

20.1. KDUMP システムロールの変数

以下のロール変数を使用して、複数のシステムに RHEL システムロールのカーネルダンプパラメーターを設定します。

ロール変数	説明
kdump_target	クラッシュダンプファイル (**vmcore**) をルートファイルシステム以外の場所に保存するために、ターゲットの場所を指定するオプション。タイプが **raw** または **filesystem** の場合、ターゲットの場所は、デバイスノード名、**label**、または **uuid** などのパーティションを参照します。
kdump_path	**vmcore** が書き込まれるパス。**kdump_target** が null ではない場合、パスはそのダンプターゲットへの相対パスになります。そうでない場合は、root ファイルシステムの絶対パスである必要があります。
kdump_core_collector	クラッシュダンプ (**vmcore**) ファイルをコピーするコマンド。null の場合、**kdump** は **kdump_target.type** に依存するオプションを指定して **makedumpfile** プログラムを使用します。
kdump_system_action	**kdump** がコアダンプファイル (**vmcore**) をプライマリーターゲットに保存できない場合に実行する代替操作。追加の操作には、**reboot**、**halt**、**poweroff**、および **shell** が含まれます。
kdump_auto_reset_crashkernel	**crashkernel** 値を新しいデフォルト値にリセットするオプション。たとえば、**kexec-tools** がデフォルトの **crashkernel** 値を新しい値に更新する場合、または既存のカーネルが古いデフォルトのカーネルの **crashkernel** 値を持っている場合は、**crashkernel** をリセットします。
kdump_dracut_args	**kdump initrd** を再構築するときに追加の **dracut** オプションを渡すオプション。

ロール変数	説明
kdump_reboot_ok	クラッシュカーネル用に十分なメモリーが予約されていない管理対象ノードでロールを実行する場合に **reboot** アクションを設定するオプション。たとえば、ファイル **/sys/kernel/kexec_crash_size** に含まれるクラッシュサイズが **0** の場合、管理対象ノードを再起動して、**kdump** を再度設定しなければならない可能性があります。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.kdump/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/kdump/** ディレクトリー
- **makedumpfile(8)** の man ページ

20.2. RHEL システムロールを使用した KDUMP の設定

kdump システムロールを使用すると、Ansible Playbook を実行して複数のシステムに基本的なカーネルダンプパラメーターを設定できます。

警告

kdump システムロールは、**/etc/kdump.conf** ファイルを置き換えて、管理対象ホストの **kdump** 設定をすべて置き換えます。また、**kdump** ロールが適用されると、**/etc/sysconfig/kdump** ファイルを置き換えて、ロール変数で指定されていない場合でも、以前の **kdump** の設定もすべて置き換えられます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- hosts: managed-node-01.example.com
  roles:
```

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

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.kdump/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/kdump/** ディレクトリー

第21章 RHEL システムロールを使用したローカルストレージの管理

Ansible を使用して LVM とローカルファイルシステム (FS) を管理するには、RHEL 9 で使用可能な RHEL システムロールの 1 つである **storage** ロールを使用できます。

storage ロールを使用すると、ディスク上のファイルシステム、複数のマシンにある論理ボリューム、および RHEL 7.7 以降の全バージョンでのファイルシステムの管理を自動化できます。

21.1. STORAGE RHEL システムロールの概要

storage ロールは以下を管理できます。

- パーティションが分割されていないディスクのファイルシステム
- 論理ボリュームとファイルシステムを含む完全な LVM ボリュームグループ
- MD RAID ボリュームとそのファイルシステム

storage ロールを使用すると、次のタスクを実行できます。

- ファイルシステムを作成する
- ファイルシステムを削除する
- ファイルシステムをマウントする
- ファイルシステムをアンマウントする
- LVM ボリュームグループを作成する
- LVM ボリュームグループを削除する
- 論理ボリュームを作成する
- 論理ボリュームを削除する
- RAID ボリュームを作成する
- RAID ボリュームを削除する
- RAID で LVM ボリュームグループを作成する
- RAID で LVM ボリュームグループを削除する
- 暗号化された LVM ボリュームグループを作成する
- RAID で LVM 論理ボリュームを作成する

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

21.2. STORAGE RHEL システムロールのストレージデバイスを識別するパラメーター

storage ロールの設定は、次の変数に指定したファイルシステム、ボリューム、およびプールにのみ適用されます。

storage_volumes

マネージドのパーティションが分割されていない全ディスク上のファイルシステムのリスト
storage_volumes には **raid** ボリュームを含めることもできます。

現在、パーティションはサポートされていません。

storage_pools

管理するプールのリスト
現在、サポートされている唯一のプールタイプは LVM です。LVM では、プールはボリュームグループ (VG) を表します。各プールの下には、ロールで管理されるボリュームのリストがあります。
LVM では、各ボリュームは、ファイルシステムを持つ論理ボリューム (LV) に対応します。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

21.3. STORAGE RHEL システムロールを使用してブロックデバイスに XFS ファイルシステムを作成する

Ansible Playbook の例では、**storage** ロールを適用して、デフォルトのパラメーターを使用してブロックデバイス上に XFS ファイルシステムを作成します。

注記

storage ロールは、パーティションが分割されていないディスク全体または論理ボリューム (LV) でのみファイルシステムを作成できます。パーティションにファイルシステムを作成することはできません。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- hosts: managed-node-01.example.com
  roles:
```

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

- 現在、ボリューム名 (この例では **barefs**) は任意です。**storage** ロールは、**disks:** 属性にリスト表示されているディスクデバイスでボリュームを特定します。
- XFS は RHEL 9 のデフォルトファイルシステムであるため、**fs_type: xfs** 行を省略することができます。
- 論理ボリュームにファイルシステムを作成するには、エンクロージングボリュームグループを含む **disks:** 属性の下に LVM 設定を指定します。詳細は、storage RHEL システムロールを使用して論理ボリュームを管理する を参照してください。
  LV デバイスへのパスを指定しないでください。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

21.4. STORAGE RHEL システムロールを使用してファイルシステムを永続的にマウントする

Ansible の例では、**storage** ロールを適用して、XFS ファイルシステムを即時かつ永続的にマウントします。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: xfs
        mount_point: /mnt/data
        mount_user: somebody
        mount_group: somegroup
        mount_mode: 0755
```

   - この Playbook は、ファイルシステムを **/etc/fstab** ファイルに追加し、ファイルシステムを即座にマウントします。
   - **/dev/sdb** デバイス上のファイルシステム、またはマウントポイントのディレクトリーが存在しない場合は、Playbook により作成されます。

2. Playbook の構文を検証します。

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

   このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

21.5. STORAGE RHEL システムロールを使用して論理ボリュームを管理する

Ansible Playbook の例では、**storage** ロールを適用して、ボリュームグループに LVM 論理ボリュームを作成します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。

- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_pools:
      - name: myvg
        disks:
          - sda
          - sdb
          - sdc
        volumes:
          - name: mylv
            size: 2G
            fs_type: ext4
            mount_point: /mnt/dat
```

   - **myvg** ボリュームグループは、ディスク **/dev/sda**、**/dev/sdb**、および **/dev/sdc** で構成されています。
   - **myvg** ボリュームグループがすでに存在する場合は、Playbook により論理ボリュームがボリュームグループに追加されます。
   - **myvg** ボリュームグループが存在しない場合は、Playbook により作成されます。
   - この Playbook は、**mylv** 論理ボリュームに Ext4 ファイルシステムを作成し、そのファイルシステムを **/mnt** に永続的にマウントします。

2. Playbook の構文を検証します。

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

   このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

21.6. STORAGE RHEL システムロールを使用してオンラインのブロック破棄を有効にする

Ansible Playbook の例では、**storage** ロールを適用して、オンラインのブロック破棄を有効にして XFS ファイルシステムをマウントします。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: xfs
        mount_point: /mnt/data
        mount_options: discard
```

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

21.7. STORAGE RHEL システムロールを使用して EXT4 ファイルシステムを作成およびマウントする

Ansible Playbook の例では、**storage** ロールを適用して、Ext4 ファイルシステムを作成してマウントします。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: ext4
        fs_label: label-name
        mount_point: /mnt/data
```

   - この Playbook は、/**dev**/**sdb** ディスクにファイルシステムを作成します。
   - この Playbook は、ファイルシステムを /**mnt**/**data** ディレクトリーに永続的にマウントします。
   - ファイルシステムのラベルは **label-name** です。

2. Playbook の構文を検証します。

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

   このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- /**usr**/**share**/**ansible**/**roles**/**rhel-system-roles.storage**/**README.md** ファイル
- /**usr**/**share**/**doc**/**rhel-system-roles**/**storage**/ ディレクトリー

21.8. STORAGE RHEL システムロールを使用して EXT3 ファイルシステムを作成およびマウントする

Ansible Playbook の例では、**storage** ロールを適用して Ext3 ファイルシステムを作成してマウントします。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- hosts: all
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: ext3
        fs_label: label-name
        mount_point: /mnt/data
        mount_user: somebody
        mount_group: somegroup
        mount_mode: 0755
```

   - この Playbook は、/**dev**/**sdb** ディスクにファイルシステムを作成します。
   - この Playbook は、ファイルシステムを /**mnt**/**data** ディレクトリーに永続的にマウントします。
   - ファイルシステムのラベルは **label-name** です。

2. Playbook の構文を検証します。

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

   このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- /**usr**/**share**/**ansible**/**roles**/**rhel-system-roles.storage**/**README.md** ファイル

- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

21.9. STORAGE RHEL システムロールを使用して LVM 上の既存のファイルシステムのサイズを変更する

このサンプル Ansible Playbook は、**storage** RHEL システムロールを適用して、ファイルシステムを持つ LVM 論理ボリュームのサイズを変更します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Create LVM pool over three disks
  hosts: managed-node-01.example.com
  tasks:
    - name: Resize LVM logical volume with file system
      ansible.builtin.include_role:
        name: rhel-system-roles.storage
      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
```

この Playbook は、以下の既存のファイルシステムのサイズを変更します。

- **/opt/mount1** にマウントされる **mylv1** ボリュームの Ext4 ファイルシステムは、そのサイズを 10 GiB に変更します。
- **/opt/mount2** にマウントされる **mylv2** ボリュームの Ext4 ファイルシステムは、そのサイズを 50 GiB に変更します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

21.10. STORAGE RHEL システムロールを使用してスワップボリュームを作成する

このセクションでは、Ansible Playbook の例を示します。この Playbook は、**storage** ロールを適用し、デフォルトのパラメーターを使用して、ブロックデバイスにスワップボリュームが存在しない場合は作成し、スワップボリュームがすでに存在する場合はそれを変更します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- name: Create a disk device with swap
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: swap_fs
        type: disk
        disks:
          - /dev/sdb
        size: 15 GiB
        fs_type: swap
```

現在、ボリューム名 (この例では **swap_fs**) は任意です。**storage** ロールは、**disks:** 属性にリスト表示されているディスクデバイスでボリュームを特定します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

21.11. STORAGE システムロールを使用して RAID ボリュームを設定する

storage システムロールを使用すると、Red Hat Ansible Automation Platform と Ansible-Core を使用して RHEL に RAID ボリュームを設定できます。要件に合わせて RAID ボリュームを設定するためのパラメーターを使用して、Ansible Playbook を作成します。

警告

特定の状況でデバイス名が変更する場合があります。たとえば、新しいディスクをシステムに追加するときなどです。したがって、データの損失を防ぐために、Playbook で特定のディスク名を使用しないでください。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- name: Configure the storage
  hosts: managed-node-01.example.com
  tasks:
    - name: Create a RAID on sdd, sde, sdf, and sdg
      ansible.builtin.include_role:
        name: rhel-system-roles.storage
      vars:
        storage_safe_mode: false
        storage_volumes:
```

```
    - name: data
      type: raid
      disks: [sdd, sde, sdf, sdg]
      raid_level: raid0
      raid_chunk_size: 32 KiB
      mount_point: /mnt/data
      state: present
```

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

21.12. STORAGE RHEL システムロールを使用して RAID を備えた LVM プールを設定する

storage システムロールを使用すると、Red Hat Ansible Automation Platform を使用して、RAID を備えた LVM プールを RHEL に設定できます。利用可能なパラメーターを使用して Ansible Playbook を設定し、RAID を備えた LVM プールを設定できます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- name: Configure LVM pool with RAID
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_safe_mode: false
    storage_pools:
      - name: my_pool
```

```
    type: lvm
    disks: [sdh, sdi]
    raid_level: raid1
    volumes:
      - name: my_volume
        size: "1 GiB"
        mount_point: "/mnt/app/shared"
        fs_type: xfs
        state: present
```

RAID を備えた LVM プールを作成するには、**raid_level** パラメーターを使用して RAID タイプを指定する必要があります。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー
- RAID の管理

21.13. STORAGE RHEL システムロールを使用して RAID LVM ボリュームのストライプサイズを設定する

storage システムロールを使用すると、Red Hat Ansible Automation Platform を使用して、RHEL の RAID LVM ボリュームのストライプサイズを設定できます。利用可能なパラメーターを使用して Ansible Playbook を設定し、RAID を備えた LVM プールを設定できます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- name: Configure stripe size for RAID LVM volumes
```

```
hosts: managed-node-01.example.com
roles:
  - rhel-system-roles.storage
vars:
  storage_safe_mode: false
  storage_pools:
    - name: my_pool
      type: lvm
      disks: [sdh, sdi]
      volumes:
        - name: my_volume
          size: "1 GiB"
          mount_point: "/mnt/app/shared"
          fs_type: xfs
          raid_level: raid1
          raid_stripe_size: "256 KiB"
          state: present
```

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

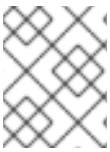
```
$ ansible-playbook ~/playbook.yml
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー
- RAID の管理

21.14. STORAGE RHEL システムロールを使用して LVM 上の VDO ボリュームを圧縮および重複排除する

このサンプル Ansible Playbook は、**storage** RHEL システムロールを適用し、Virtual Data Optimizer (VDO) を使用した論理ボリューム (LVM) の圧縮と重複排除を有効にします。



注記

storage システムロールが LVM VDO を使用するため、圧縮と重複排除を使用できるのはプールごとに 1 つのボリュームのみです。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。

- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
- name: Create LVM VDO volume under volume group 'myvg'
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_pools:
      - name: myvg
        disks:
          - /dev/sdb
        volumes:
          - name: mylv1
            compression: true
            deduplication: true
            vdo_pool_size: 10 GiB
            size: 30 GiB
            mount_point: /mnt/app/shared
```

この例では、**compression** プールおよび **deduplication** プールを true に設定します。これは、VDO が使用されることを指定します。以下では、このパラメーターの使用方法を説明します。

- **deduplication** は、ストレージボリュームに保存されている重複データの重複排除に使用されます。
- 圧縮は、ストレージボリュームに保存されているデータを圧縮するために使用されます。これにより、より大きなストレージ容量が得られます。
- vdo_pool_size は、ボリュームがデバイスで使用する実際のサイズを指定します。VDO ボリュームの仮想サイズは、**size** パラメーターで設定します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

21.15. STORAGE RHEL システムロールを使用して LUKS2 暗号化ボリュームを作成する

storage ロールを使用し、Ansible Playbook を実行して、LUKS で暗号化されたボリュームを作成および設定できます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Create and configure a volume encrypted with LUKS
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
        - sdb
        fs_type: xfs
        fs_label: label-name
        mount_point: /mnt/data
        encryption: true
        encryption_password: <password>
```

また、**encryption_key**、**encryption_cipher**、**encryption_key_size**、**encryption_luks** など、他の暗号化パラメーターを Playbook ファイルに追加することもできます。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

1. 暗号化ステータスを表示します。

```
# cryptsetup status sdb

/dev/mapper/sdb is active and is in use.
type: LUKS2
cipher: aes-xts-plain64
keysize: 512 bits
key location: keyring
device: /dev/sdb
...
```

2. 作成された LUKS 暗号化ボリュームを確認します。

```
# cryptsetup luksDump /dev/sdb

Version:        2
Epoch:          6
Metadata area:  16384 [bytes]
Keyslots area:  33521664 [bytes]
UUID:           a4c6be82-7347-4a91-a8ad-9479b72c9426
Label:          (no label)
Subsystem:      (no subsystem)
Flags:          allow-discards

Data segments:
  0: crypt
        offset: 33554432 [bytes]
        length: (whole device)
        cipher: aes-xts-plain64
        sector: 4096 [bytes]
...
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー
- LUKS を使用したブロックデバイスの暗号化

21.16. STORAGE RHEL システムロールを使用してプールボリュームのサイズをパーセンテージで表す

このサンプル Ansible Playbook は、**storage** システムロールを適用して、論理マネージャーボリューム (LVM) のボリュームサイズをプールの合計サイズのパーセンテージで表現できるようにします。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- name: Express volume sizes as a percentage of the pool's total size
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_pools:
      - name: myvg
        disks:
          - /dev/sdb
        volumes:
          - name: data
            size: 60%
            mount_point: /opt/mount/data
          - name: web
            size: 30%
            mount_point: /opt/mount/web
          - name: cache
            size: 10%
            mount_point: /opt/cache/mount
```

この例では、LVM ボリュームのサイズをプールサイズのパーセンテージで指定します (例: **60%**)。LVM ボリュームのサイズは、人間が判読できるファイルシステムのサイズ (例: **10g** または **50 GiB**) に占めるプールサイズのパーセンテージで指定することもできます。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

第22章 TIMESYNC RHEL システムロールを使用した時刻同期の設定

timesync RHEL システムロールを使用すると、Red Hat Ansible Automation Platform を使用して、RHEL の複数のターゲットマシンで時刻同期を管理できます。

22.1. TIMESYNC RHEL システムロール

timesync RHEL システムロールを使用して、複数のターゲットマシンで時刻同期を管理できます。

システムクロックが NTP サーバーまたは PTP ドメインのグランドマスターに同期するように、**timesync** ロールが NTP 実装または PTP 実装をインストールし、NTP クライアントまたは PTP レプリカとして動作するように設定します。

timesync ロールを使用すると、システムが NTP プロトコルの実装に **ntp** と **chrony** のどちらを使用するかにかかわらず、RHEL 6 以降のすべてのバージョンの Red Hat Enterprise Linux で同じ Playbook を使用できるため、chrony への移行 が容易になります。

- **/usr/share/ansible/roles/rhel-system-roles.timesync/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/timesync/** ディレクトリー

22.2. TIMESYNC システムロールの変数

以下の変数を **timesync** ロールに渡すことができます。

- **timesync_ntp_servers**:

ロール変数の設定	説明
hostname: host.example.com	サーバーのホスト名またはアドレス。
minpoll: **number**	最小ポーリング間隔。デフォルト: 6
maxpoll: **number**	最大ポーリングの間隔。デフォルト: 10
iburst: yes	高速な初期同期を有効にするフラグ。デフォルト: no
pool: yes	ホスト名の解決された各アドレスが別の NTP サーバーであることを示すフラグ。デフォルト: no
nts: yes	Network Time Security (NTS) を有効にするフラグ。デフォルト: no. Supported only with chrony >= 4.0.

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.timesync/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/timesync/** ディレクトリー

22.3. 1 つのサーバープールに TIMESYNC システムロールを適用する

以下の例は、サーバーにプールが 1 つしかない場合に、**timesync** ロールを適用する方法を示しています。

警告

timesync ロールは、マネージドホストで指定または検出されたプロバイダーサービスの設定を置き換えます。以前の設定は、ロール変数で指定されていなくても失われます。**timesync_ntp_provider** 変数が定義されていない場合は、プロバイダーの唯一の設定が適用されます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Manage time synchronization
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.timesync
  vars:
    timesync_ntp_servers:
      - hostname: 2.rhel.pool.ntp.org
        pool: yes
        iburst: yes
```

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.timesync/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/timesync/** ディレクトリー

22.4. クライアントサーバーに TIMESYNC システムロールを適用する

timesync ロールを使用すると、NTP クライアントで Network Time Security (NTS) を有効にできます。Network Time Security (NTS) は、Network Time Protocol (NTP) で指定される認証メカニズムです。サーバーとクライアント間で交換される NTP パケットが変更されていないことを確認します。

警告

timesync ロールは、マネージドホストで指定または検出されたプロバイダーサービスの設定を置き換えます。以前の設定は、ロール変数で指定されていなくても失われます。**timesync_ntp_provider** 変数が定義されていない場合は、プロバイダーの唯一の設定が適用されます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- **chrony** の NTP プロバイダーバージョンは 4.0 以降。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Enable Network Time Security on NTP clients
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.timesync
  vars:
    timesync_ntp_servers:
      - hostname: ptbtime1.ptb.de
        iburst: yes
        nts: yes
```

ptbtime1.ptb.de はパブリックサーバーの例です。別のパブリックサーバーや独自のサーバーを使用することもできます。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

1. クライアントマシンでテストを実行します。

```
# chronyc -N authdata

Name/IP address         Mode KeyID Type KLen Last Atmp  NAK Cook CLen
==============================================================================
ptbtime1.ptb.de         NTS     1   15  256  157    0    0    8  100
```

2. 報告された cookie の数がゼロよりも多いことを確認します。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.timesync/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/timesync/** ディレクトリー

第23章 METRICS RHEL システムロールを使用したパフォーマンスの監視

システム管理者は、**metrics** RHEL システムロールを使用して、システムのパフォーマンスを監視できます。

23.1. METRICS システムロールの概要

RHEL システムロールは、複数の RHEL システムをリモートで管理するための一貫した設定インターフェイスを提供する Ansible ロールおよびモジュールのコレクションです。**metrics** システムロールは、ローカルシステムのパフォーマンス分析サービスを設定します。必要に応じて、ローカルシステムによって監視される一連のリモートシステムも分析の対象とします。**metrics** システムロールを使用すると、**pcp** のセットアップとデプロイが Playbook によって処理されるため、**pcp** を別途設定しなくても、**pcp** を使用してシステムのパフォーマンスを監視できます。

表23.1metrics システムロール変数

ロール変数	説明	使用例
metrics_monitored_hosts	ターゲットホストが分析するリモートホストのリスト。これらのホストにはターゲットホストにメトリックが記録されるため、各ホストの **/var/log** の下に十分なディスク領域があることを確認してください。	**metrics_monitored_hosts: ["webserver.example.com", "database.example.com"]**
metrics_retention_days	削除前のパフォーマンスデータの保持日数を設定します。	**metrics_retention_days: 14**
metrics_graph_service	**pcp** および **grafana** を介してパフォーマンスデータの視覚化のためにホストをサービスで設定できるようにするブール値フラグ。デフォルトでは false に設定されます。	**metrics_graph_service: no**
metrics_query_service	**redis** 経由で記録された **pcp** メトリックをクエリーするための時系列クエリーサービスでのホストの設定を可能にするブール値フラグ。デフォルトでは false に設定されます。	**metrics_query_service: no**
metrics_provider	メトリックを提供するために使用するメトリックコレクターを指定します。現在、サポートされている唯一のメトリックプロバイダーは **pcp** です。	**metrics_provider: "pcp"**

ロール変数	説明	使用例
metrics_manage_firewall	**firewall** ロールを使用して、**metrics** ロールからポートアクセスを直接管理します。デフォルトでは false に設定されます。	**metrics_manage_firewall: true**
metrics_manage_selinux	**selinux** ロールを使用して、**metrics** ロールから直接ポートアクセスを管理します。デフォルトでは false に設定されます。	**metrics_manage_selinux: true**

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/metrics/** ディレクトリー

23.2. METRICS システムロールを使用して視覚的にローカルシステムを監視する

この手順では、**metrics** RHEL システムロールを使用してローカルシステムを監視しながら、同時に **Grafana** によるデータの視覚化をプロビジョニングする方法について説明します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- **localhost** がコントロールノードのインベントリーファイルで設定されている。

```
localhost ansible_connection=local
```

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Manage metrics
  hosts: localhost
  roles:
    - rhel-system-roles.metrics
  vars:
    metrics_graph_service: yes
    metrics_manage_firewall: true
    metrics_manage_selinux: true
```

metrics_graph_service のブール値が **value="yes"** に設定されているため、**Grafana** は自動的

にインストールされ、データソースとして追加された **pcp** でプロビジョニングされます。**metrics_manage_firewall** と **metrics_manage_selinux** は両方とも **true** に設定されているため、メトリクスロールは **firewall** および **selinux** システムロールを使用して、メトリクスロールが使用するポートを管理します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

- マシンで収集されるメトリクスを視覚化するには、Grafana Web UI へのアクセス の説明どおりに **grafana** Web インターフェイスにアクセスします。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/metrics/** ディレクトリー

23.3. METRICS システムロールを使用して自己監視するようにシステム群を設定する

この手順では、**metrics** システムロールを使用して、マシン群が自己監視するように設定する方法を説明します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- name: Configure a fleet of machines to monitor themselves
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.metrics
  vars:
```

```
    metrics_retention_days: 0
    metrics_manage_firewall: true
    metrics_manage_selinux: true
```

metrics_manage_firewall と **metrics_manage_selinux** は両方とも **true** に設定されているため、メトリクスロールは **firewall** ロールと **selinux** ロールを使用して、**metrics** ロールが使用するポートを管理します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/metrics/** ディレクトリー

23.4. METRICS システムロールを使用してローカルマシンからマシン群を一元的に監視する

この手順では、**メトリクス** システムロールを使用してローカルマシンを設定し、マシン群を一元的に監視するとともに、**Grafana** によるデータの視覚化をプロビジョニングし、**Redis** によりデータをクエリーする方法について説明します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- **localhost** がコントロールノードのインベントリーファイルで設定されている。

```
localhost ansible_connection=local
```

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
- name: Set up your local machine to centrally monitor a fleet of machines
  hosts: localhost
  roles:
    - rhel-system-roles.metrics
```

```
vars:
  metrics_graph_service: yes
  metrics_query_service: yes
  metrics_retention_days: 10
  metrics_monitored_hosts: ["database.example.com", "webserver.example.com"]
  metrics_manage_firewall: yes
  metrics_manage_selinux: yes
```

metrics_graph_service および **metrics_query_service** のブール値は **value="yes"** に設定されているため、**grafana** は、**redis** にインデックス化された **pcp** データの記録のあるデータソースとして追加された **pcp** で自動的にインストールおよびプロビジョニングされます。これにより、**pcp** クエリー言語をデータの複雑なクエリーに使用できます。**metrics_manage_firewall** と **metrics_manage_selinux** は両方とも **true** に設定されているため、**metrics** ロールは **firewall** ロールと **selinux** ロールを使用して、**metrics** ロールが使用するポートを管理します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

- マシンによって一元的に収集されるメトリクスのグラフィック表示とデータのクエリーを行うには、Grafana Web UI へのアクセス で説明されているように、**grafana** Web インターフェイスにアクセスします。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/metrics/** ディレクトリー

23.5. METRICS システムロールを使用してシステムを監視しながら認証を設定する

PCP は、Simple Authentication Security Layer (SASL) フレームワークを介して **scram-sha-256** 認証メカニズムに対応します。**metrics** RHEL システムロールは、**scram-sha-256** 認証メカニズムを使用して認証を設定する手順を自動化します。この手順では、**metrics** RHEL システムロールを使用して認証を設定する方法について説明します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。

- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 既存の Playbook ファイル (例: ~/**playbook.yml)** を編集し、認証関連の変数を追加します。

```
---
- name: Set up authentication by using the scram-sha-256 authentication mechanism
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.metrics
  vars:
    metrics_retention_days: 0
    metrics_manage_firewall: true
    metrics_manage_selinux: true
    metrics_username: <username>
    metrics_password: <password>
```

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

- **sasl** 設定を確認します。

```
# pminfo -f -h "pcp://managed-node-01.example.com?username=<username>"
disk.dev.read
Password: <password>
disk.dev.read
inst [0 or "sda"] value 19540
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/metrics/** ディレクトリー

23.6. METRICS システムロールを使用して SQL SERVER のメトリクス収集を設定して有効にする

この手順では、**metrics** RHEL システムロールを使用して、ローカルシステムでの **pcp** を使用した Microsoft SQL Server のメトリクス収集の設定と有効化を自動化する方法について説明します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- Red Hat Enterprise Linux 用の Microsoft SQL Server をインストールし、SQL サーバーへの信頼できる接続が確立されている。
- Red Hat Enterprise Linux 用の SQL Server の Microsoft ODBC ドライバーがインストールされている。
- **localhost** がコントロールノードのインベントリーファイルで設定されている。

```
localhost ansible_connection=local
```

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Configure and enable metrics collection for Microsoft SQL Server
  hosts: localhost
  roles:
    - rhel-system-roles.metrics
  vars:
    metrics_from_mssql: true
    metrics_manage_firewall: true
    metrics_manage_selinux: true
```

metrics_manage_firewall と **metrics_manage_selinux** は両方とも **true** に設定されているため、**metrics** ロールは **firewall** ロールと **selinux** ロールを使用して、**metrics** ロールが使用するポートを管理します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

- **pcp** コマンドを使用して、SQL Server PMDA エージェント (mssql) が読み込まれ、実行されていることを確認します。

```
# pcp
platform: Linux sqlserver.example.com 4.18.0-167.el8.x86_64 #1 SMP Sun Dec 15 01:24:23
UTC 2019 x86_64
```

```
hardware: 2 cpus, 1 disk, 1 node, 2770MB RAM
timezone: PDT+7
services: pmcd pmproxy
    pmcd: Version 5.0.2-1, 12 agents, 4 clients
    pmda: root pmcd proc pmproxy xfs linux nfsclient mmv kvm mssql
          jbd2 dm
pmlogger: primary logger: /var/log/pcp/pmlogger/sqlserver.example.com/20200326.16.31
    pmie: primary engine: /var/log/pcp/pmie/sqlserver.example.com/pmie.log
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/metrics/** ディレクトリー

第24章 TLOG RHEL システムロールを使用したセッション記録用システムの設定

tlog RHEL システムロールを使用すると、Red Hat Ansible Automation Platform を使用して、RHEL でターミナルセッションを記録するようにシステムを設定できます。

24.1. TLOG システムロール

tlog RHEL システムロールを使用して、RHEL でターミナルセッションを記録するように RHEL システムを設定できます。

SSSD サービスを使用して、ユーザーまたはユーザーグループごとに記録を行うように設定できます。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/ha_cluster/** ディレクトリー

24.2. TLOG システムロールのコンポーネントとパラメーター

セッション記録ソリューションには次のコンポーネントがあります。

- **tlog** ユーティリティー
- System Security Services Daemon (SSSD)
- オプション: Web コンソールインターフェイス

tlog RHEL システムロールに使用されるパラメーターは次のとおりです。

ロール変数	説明
tlog_use_sssd (デフォルト: yes)	SSSD を使用してセッションの記録を設定します。記録するユーザーやグループを管理する方法として推奨されます。
tlog_scope_sssd (デフォルト: none)	SSSD の記録スコープの設定 - all / some / none
tlog_users_sssd (デフォルト: [])	記録するユーザーの YAML リスト
tlog_groups_sssd (デフォルト: [])	記録するグループの YAML リスト

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/ha_cluster/** ディレクトリー

24.3. TLOG RHEL システムロールのデプロイ

次の手順に従って、Ansible Playbook を準備して適用し、セッション記録データを systemd ジャーナルに記録するように RHEL システムを設定します。

この Playbook は、指定されたシステムに **tlog** RHEL システムロールをインストールします。このロールには、ユーザーのログインシェルとして機能するターミナルセッション I/O ロギングプログラムである **tlog-rec-session** が含まれます。また、定義したユーザーおよびグループで使用できる SSSD 設定ドロップファイルを作成します。SSSD は、これらのユーザーとグループを解析して読み取り、ユーザーシェルを **tlog-rec-session** に置き換えます。さらに、**cockpit** パッケージがシステムにインストールされている場合、Playbook は **cockpit-session-recording** パッケージもインストールします。これは **Cockpit** モジュールの 1 つであり、Web コンソールインターフェイスでの記録の表示と再生を可能にするものです。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Deploy session recording
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.tlog
  vars:
    tlog_scope_sssd: some
    tlog_users_sssd:
      - recorded-user
```

tlog_scope_sssd

some 値は、**all** または **none** ではなく、特定のユーザーとグループのみを記録することを指定します。

tlog_users_sssd

セッションを記録するユーザーを指定します。ただし、ユーザーは追加されない点に留意してください。ユーザーを独自に設定する必要があります。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

1. SSSD 設定ドロップファイルが作成されるフォルダーに移動します。

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

2. ファイルの内容を確認します。

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

Playbook に設定したパラメーターがファイルに含まれていることが確認できます。

3. セッションを記録するユーザーとしてログインします。
4. 記録されたセッションを再生します。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.tlog/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/tlog/** ディレクトリー

24.4. グループまたはユーザーのリストを除外するために TLOG RHEL システムロールをデプロイする

tlog システムロールを使用すると、SSSD セッションの録画設定オプション **exclude_users** および **exclude_groups** をサポートできます。次の手順に従って、Ansible Playbook を準備して適用し、ユーザーまたはグループのセッションが systemd ジャーナルに記録およびログされないように RHEL システムを設定します。

この Playbook は、指定されたシステムに **tlog** RHEL システムロールをインストールします。このロールには、ユーザーのログインシェルとして機能するターミナルセッション I/O ロギングプログラムである **tlog-rec-session** が含まれます。また、除外対象外のユーザーおよびグループが使用できる **/etc/sssd/conf.d/sssd-session-recording.conf** SSSD 設定ドロップファイルを作成します。SSSD は、これらのユーザーとグループを解析して読み取り、ユーザーシェルを **tlog-rec-session** に置き換えます。さらに、**cockpit** パッケージがシステムにインストールされている場合、Playbook は **cockpit-session-recording** パッケージもインストールします。これは **Cockpit** モジュールの 1 つであり、Web コンソールインターフェイスでの記録の表示と再生を可能にするものです。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Deploy session recording excluding users and groups
  hosts: managed-node-01.example.com
  roles:
```

```
    - rhel-system-roles.tlog
  vars:
    tlog_scope_sssd: all
    tlog_exclude_users_sssd:
      - jeff
      - james
    tlog_exclude_groups_sssd:
      - admins
```

tlog_scope_sssd

値 **all** は、すべてのユーザーとグループを記録することを指定します。

tlog_exclude_users_sssd

セッションの記録から除外するユーザーのユーザー名を指定します。

tlog_exclude_groups_sssd

セッション記録から除外するグループを指定します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

1. SSSD 設定ドロップファイルが作成されるフォルダーに移動します。

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

2. ファイルの内容を確認します。

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

Playbook に設定したパラメーターがファイルに含まれていることが確認できます。

3. セッションを記録するユーザーとしてログインします。

4. 記録されたセッションを再生します。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.tlog/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/tlog/** ディレクトリー

第25章 HA_CLUSTER RHEL システムロールを使用した高可用性クラスターの設定

ha_cluster システムロールを使用すると、Pacemaker の高可用性クラスターリソースマネージャーを使用する高可用性クラスターを設定し、管理できます。

25.1. HA_CLUSTER システムロールの変数

ha_cluster システムロール Playbook では、クラスターデプロイメントの要件に従って、高可用性クラスターの変数を定義します。

ha_cluster システムロールに設定できる変数は次のとおりです。

ha_cluster_enable_repos

ha_cluster システムロールで必要なパッケージを含むリポジトリーを有効にするブール値フラグ。この変数がデフォルト値の **true** に設定されている場合、クラスターメンバーとして使用するシステムに RHEL および RHEL High Availability Add-On の有効なサブスクリプションが必要です。サブスクリプションがない場合、システムロールは失敗します。

ha_cluster_manage_firewall

(RHEL 9.2 以降) **ha_cluster** システムロールがファイアウォールを管理するかどうかを決定するブール値フラグ。**ha_cluster_manage_firewall** が **true** に設定されている場合、ファイアウォールの高可用性サービスと **fence-virt** ポートが有効になります。**ha_cluster_manage_firewall** が **false** に設定されている場合、**ha_cluster** システムロールはファイアウォールを管理しません。システムが **firewalld** サービスを実行している場合は、Playbook でパラメーターを **true** に設定する必要があります。
ha_cluster_manage_firewall パラメーターを使用してポートを追加することはできますが、このパラメーターを使用してポートを削除することはできません。ポートを削除するには、**firewall** システムロールを直接使用します。

RHEL 9.2 以降、ファイアウォールはデフォルトでは設定されなくなりました。これは、ファイアウォールが設定されるのは、**ha_cluster_manage_firewall** が **true** に設定されている場合のみであるためです。

ha_cluster_manage_selinux

(RHEL 9.2 以降) **ha_cluster** システムロールが **selinux** システムロールを使用してファイアウォール高可用性サービスに属するポートを管理するかどうかを決定するブール値フラグ。**ha_cluster_manage_selinux** が **true** に設定されている場合、ファイアウォール高可用性サービスに属するポートは、SELinux ポートタイプ **cluster_port_t** に関連付けられます。**ha_cluster_manage_selinux** が **false** に設定されている場合、**ha_cluster** システムロールは SELinux を管理しません。
システムが **selinux** サービスを実行している場合は、Playbook でこのパラメーターを **true** に設定する必要があります。ファイアウォール設定は、SELinux を管理するための前提条件です。ファイアウォールがインストールされていない場合、SELinux ポリシーの管理はスキップされます。

ha_cluster_manage_selinux パラメーターを使用してポリシーを追加することはできますが、このパラメーターを使用してポリシーを削除することはできません。ポリシーを削除するには、**selinux** システムロールを直接使用します。

ha_cluster_cluster_present

true に設定すると、ロールに渡された変数に従って HA クラスターがホスト上で設定されることを決定するブール型フラグ。ロールで指定されておらず、ロールによってサポートされないクラスター設定は失われます。

ha_cluster_cluster_present を **false** に設定すると、すべての HA クラスター設定がターゲットホストから削除されます。

この変数のデフォルト値は **true** です。

以下の Playbook の例では、**node1** および **node2** のすべてのクラスター設定を削除します。

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

  roles:
    - rhel-system-roles.ha_cluster
```

ha_cluster_start_on_boot

起動時にクラスターサービスが起動するように設定されるかどうかを決定するブール値フラグ。この変数のデフォルト値は **true** です。

ha_cluster_fence_agent_packages

インストールするフェンスエージェントパッケージのリストこの変数のデフォルト値は **fence-agents-all**, **fence-virt** です。

ha_cluster_extra_packages

インストールする追加パッケージのリストこの変数のデフォルト値はパッケージではありません。この変数は、ロールによって自動的にインストールされていない追加パッケージをインストールするために使用できます (例: カスタムリソースエージェント)。

フェンスエージェントをこのリストのメンバーとして追加できます。ただし、**ha_cluster_fence_agent_packages** は、フェンスエージェントの指定に使用する推奨されるロール変数であるため、デフォルト値が上書きされます。

ha_cluster_hacluster_password

hacluster ユーザーのパスワードを指定する文字列の値。**hacluster** ユーザーには、クラスターへの完全アクセスがあります。機密データを保護するには、Encrypting content with Ansible Vault で説明されているように、パスワードを vault で暗号化します。デフォルトのパスワード値がないため、この変数を指定する必要があります。

ha_cluster_hacluster_qdevice_password

(RHEL 9.3 以降) クォーラムデバイスの **hacluster** ユーザーのパスワードを指定する文字列の値。このパラメーターが必要になるのは、**ha_cluster_quorum** パラメーターがタイプ **net** のクォーラムデバイスを使用するように設定されており、クォーラムデバイス上の **hacluster** ユーザーのパスワードが **ha_cluster_hacluster_password** パラメーターで指定された **hacluster** ユーザーのパスワードと異なる場合のみです。**hacluster** ユーザーには、クラスターへの完全アクセスがあります。機密データを保護するには、Encrypting content with Ansible Vault で説明されているように、パスワードを vault で暗号化します。このパスワードにデフォルト値はありません。

ha_cluster_corosync_key_src

Corosync **authkey** ファイルへのパス。これは、Corosync 通信の認証および暗号鍵です。各クラスターに一意の **authkey** 値を指定することが強く推奨されます。キーは、ランダムなデータの 256 バイトでなければなりません。
この変数の鍵を指定する場合は、Ansible Vault を使用したコンテンツの暗号化 で説明されているように、鍵を vault 暗号化することが推奨されます。

鍵が指定されていない場合は、ノードにすでに存在するキーが使用されます。ノードに同じ鍵がない場合、あるノードの鍵が他のノードに分散され、すべてのノードが同じキーを持つようにします。ノードに鍵がない場合は、新しい鍵が生成され、ノードに分散されます。

この変数が設定されている場合は、このキーで **ha_cluster_regenerate_keys** が無視されます。

この変数のデフォルト値は null です。

ha_cluster_pacemaker_key_src

Pacemaker の **authkey** ファイルへのパスです。これは、Pacemaker 通信の認証および暗号鍵です。各クラスターに一意の **authkey** 値を指定することが強く推奨されます。キーは、ランダムなデータの 256 バイトでなければなりません。
この変数の鍵を指定する場合は、Ansible Vault を使用したコンテンツの暗号化 で説明されているように、鍵を vault 暗号化することが推奨されます。

鍵が指定されていない場合は、ノードにすでに存在するキーが使用されます。ノードに同じ鍵がない場合、あるノードの鍵が他のノードに分散され、すべてのノードが同じキーを持つようにします。ノードに鍵がない場合は、新しい鍵が生成され、ノードに分散されます。

この変数が設定されている場合は、このキーで **ha_cluster_regenerate_keys** が無視されます。

この変数のデフォルト値は null です。

ha_cluster_fence_virt_key_src

fence-virt または **fence-xvm** の事前共有鍵ファイルへのパス。これは、**fence-virt** または **fence-xvm** フェンスエージェントの認証キーの場所になります。
この変数の鍵を指定する場合は、Ansible Vault を使用したコンテンツの暗号化 で説明されているように、鍵を vault 暗号化することが推奨されます。

鍵が指定されていない場合は、ノードにすでに存在するキーが使用されます。ノードに同じ鍵がない場合、あるノードの鍵が他のノードに分散され、すべてのノードが同じキーを持つようにします。ノードに鍵がない場合は、新しい鍵が生成され、ノードに分散されます。この方法で **ha_cluster** システムロールが新しいキーを生成する場合は、鍵をノードのハイパーバイザーにコピーして、フェンシングが機能するようにする必要があります。

この変数が設定されている場合は、このキーで **ha_cluster_regenerate_keys** が無視されます。

この変数のデフォルト値は null です。

ha_cluster_pcsd_public_key_srcr、ha_cluster_pcsd_private_key_src

pcsd TLS 証明書および秘密鍵へのパス。これが指定されていない場合は、ノード上にすでに証明書キーのペアが使用されます。証明書キーペアが存在しない場合は、無作為に新しいキーが生成されます。
この変数に秘密鍵の値を指定した場合は、Ansible Vault を使用したコンテンツの暗号化 で説明されているように、鍵を暗号化することが推奨されます。

これらの変数が設定されている場合は、この証明書と鍵のペアで **ha_cluster_regenerate_keys** は無視されます。

これらの変数のデフォルト値は null です。

ha_cluster_pcsd_certificates

(RHEL 9.2 以降) **certificate** システムロールを使用して **pcsd** 秘密鍵と証明書を作成します。
システムに **pcsd** 秘密鍵と証明書が設定されていない場合は、次の 2 つの方法のいずれかで作成できます。

- **ha_cluster_pcsd_certificates** 変数を設定します。**ha_cluster_pcsd_certificates** 変数を設定すると、**certificate** システムロールが内部的に使用され、定義どおりに **pcsd** の秘密鍵と証明書が作成されます。

- **ha_cluster_pcsd_public_key_src**、**ha_cluster_pcsd_private_key_src**、または **ha_cluster_pcsd_certificates** 変数を設定しないでください。これらの変数をいずれも設定しなかった場合、**ha_cluster** システムロールが **pcsd** 自体を使用して **pcsd** 証明書を作成します。**ha_cluster_pcsd_certificates** の値は、**certificate** システムロールで指定された変数 **certificate_requests** の値に設定されます。**certificate** システムのロールの詳細は、RHEL システムロールを使用した証明書の要求 を参照してください。

ha_cluster_pcsd_certificate 変数の使用には、次の運用上の考慮事項が適用されます。

- IPA を使用しており、システムを IPA ドメインに参加させていない限り、**certificate** システムロールによって自己署名証明書が作成されます。この場合、RHEL システムロールのコンテキスト外で信頼設定を明示的に設定する必要があります。システムロールは、信頼設定をサポートしていません。

- **ha_cluster_pcsd_certificates** 変数を設定する場合は、**ha_cluster_pcsd_public_key_src** 変数と **ha_cluster_pcsd_private_key_src** 変数を設定しないでください。

- **ha_cluster_pcsd_certificates** 変数を設定すると、この証明書とキーのペアでは **ha_cluster_regenerate_keys** が無視されます。

この変数のデフォルト値は **[]** です。
高可用性クラスターで TLS 証明書とキーファイルを作成する **ha_cluster** システムロール Playbook の例については、高可用性クラスターの pcsd TLS 証明書とキーファイルの作成 を参照してください。

ha_cluster_regenerate_keys

true に設定すると、事前共有キーと TLS 証明書が再生成されることを決定するブール型フラグ。
キーおよび証明書が再生成されるタイミングの詳細は、**ha_cluster_corosync_key_src**、**ha_cluster_pacemaker_key_src**、**ha_cluster_fence_virt_key_src**、**ha_cluster_pcsd_public_key_src**、および **ha_cluster_pcsd_private_key_src** 変数の説明を参照してください。

この変数のデフォルト値は **false** です。

ha_cluster_pcs_permission_list

pcsd を使用してクラスターを管理するパーミッションを設定します。この変数を使用して設定するアイテムは以下のとおりです。

- **type** - **user** または **group**

- **name** - ユーザーまたはグループの名前

- **allow_list** - 指定されたユーザーまたはグループの許可されるアクション:

  - **read** - クラスターのステータスおよび設定の表示

  - **write** - パーミッションおよび ACL を除くクラスター設定の変更

  - **grant** - クラスターパーミッションおよび ACL の変更

  - **full** - ノードの追加および削除、キーおよび証明書へのアクセスなど、クラスターへの無制限アクセス

ha_cluster_pcs_permission_list 変数の構造とデフォルト値は以下のとおりです。

```
ha_cluster_pcs_permission_list:
  - type: group
    name: hacluster
    allow_list:
      - grant
      - read
      - write
```

ha_cluster_cluster_name

クラスターの名前。これは、デフォルトが **my-cluster** の文字列値です。

ha_cluster_transport

(RHEL 9.1 以降) クラスターのトランスポート方式を設定します。この変数を使用して設定するアイテムは以下のとおりです。

- **type** (オプション) - トランスポートタイプ: **knet**、**udp**、または **udpu**。**udp** および **udpu** トランスポートタイプは、1つのリンクのみをサポートします。**udp** と **udpu** の暗号化は常に無効になっています。指定しない場合、デフォルトで **knet** になります。

- **options** (オプション) - トランスポートオプションを含む名前と値のディクショナリーのリスト。

- **links** (オプション) - 名前と値のディクショナリーのリスト。名前値ディクショナリーの各リストには、1つの Corosync リンクのオプションが含まれています。リンクごとに **linknumber** 値を設定することを推奨します。それ以外の場合、ディクショナリーの最初のリストはデフォルトで最初のリンクに割り当てられ、2 番目のリストは 2 番目のリンクに割り当てられます。

- **compression** (オプション) - トランスポート圧縮を設定する名前と値のディクショナリーのリスト。**knet** トランスポートタイプでのみサポートされます。

- **crypto** (オプション) - トランスポートの暗号化を設定する名前と値のディクショナリーのリスト。デフォルトでは、暗号化は有効になっています。**knet** トランスポートタイプでのみサポートされます。

許可されているオプションのリストについては、**pcs -h cluster setup** のヘルプページ、または **pcs**(8) man ページの **cluster** セクションにある **setup** の説明を参照してください。詳細な説明については、**corosync.conf**(5) の man ページを参照してください。
ha_cluster_transport 変数の構造は次のとおりです。

```
ha_cluster_transport:
  type: knet
  options:
    - name: option1_name
      value: option1_value
    - name: option2_name
      value: option2_value
  links:
    -
      - name: option1_name
        value: option1_value
      - name: option2_name
        value: option2_value
```

```
    -
      - name: option1_name
        value: option1_value
      - name: option2_name
        value: option2_value
  compression:
    - name: option1_name
      value: option1_value
    - name: option2_name
      value: option2_value
  crypto:
    - name: option1_name
      value: option1_value
    - name: option2_name
      value: option2_value
```

トランスポート方式を設定する **ha_cluster** システムロール Playbook の例については、高可用性クラスターでの Corosync 値の設定 を参照してください。

ha_cluster_totem

(RHEL 9.1 以降) Corosync トーテムを設定します。許可されているオプションのリストについては、**pcs -h cluster setup** のヘルプページ、または **pcs**(8) man ページの **cluster** セクションにある **setup** の説明を参照してください。詳細な説明については、**corosync.conf**(5) の man ページを参照してください。
ha_cluster_totem 変数の構造は次のとおりです。

```
ha_cluster_totem:
  options:
    - name: option1_name
      value: option1_value
    - name: option2_name
      value: option2_value
```

Corosync トーテムを設定する **ha_cluster** システムロール Playbook の例については、高可用性クラスターでの Corosync 値の設定 を参照してください。

ha_cluster_quorum

(RHEL 9.1 以降) クラスタークォーラムを設定します。クラスタークォーラムには、次の項目を設定できます。

- **options** (オプション) - クォーラムを設定する名前と値のディクショナリーのリスト。許可されるオプション
  は、**auto_tie_breaker**、**last_man_standing**、**last_man_standing_window**、および **wait_for_all** です。クォーラムオプションの詳細は、**votequorum**(5) の man ページを参照してください。

- **device** (オプション) - (RHEL 9.2 以降) クォーラムデバイスを使用するようにクラスターを設定します。デフォルトでは、クォーラムデバイスは使用されません。

  - **model** (必須) - クォーラムデバイスモデルを指定します。**net** のみがサポートされています。

  - **model_options** (オプション) - 指定されたクォーラムデバイスモデルを設定する名前と値のディクショナリーのリスト。モデル **net** の場合、**host** および **algorithm** オプションを指定する必要があります。

pcs-address オプションを使用して、**qnetd** ホストに接続するためのカスタム **pcsd** アドレスとポートを設定します。このオプションを指定しない場合、ロールは **host** 上のデフォルトの **pcsd** ポートに接続します。

- **generic_options** (オプション) - モデル固有ではないクォーラムデバイスオプションを設定する名前と値のディクショナリーのリスト。
- **heuristics_options** (オプション) - クォーラムデバイスのヒューリスティックを設定する名前と値のディクショナリーのリスト。
  クォーラムデバイスオプションの詳細は、**corosync-qdevice**(8) の man ページを参照してください。一般的なオプションは **sync_timeout** と **timeout** です。モデル **net** オプションについては、**quorum.device.net** セクションを参照してください。ヒューリスティックオプションについては、**quorum.device.heuristics** セクションを参照してください。

  クォーラムデバイスの TLS 証明書を再生成するには、**ha_cluster_regenerate_keys** 変数を **true** に設定します。

ha_cluster_quorum 変数の構造は次のとおりです。

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

クラスタークォーラムを設定する **ha_cluster** システムロール Playbook の例については、高可用性クラスターでの Corosync 値の設定 を参照してください。クォーラムデバイスを使用してクラスターを設定する **ha_cluster** システムロール Playbook の例については、 クォーラムデバイスを使用した高可用性クラスターの設定 を参照してください。

ha_cluster_sbd_enabled

(RHEL 9.1 以降) クラスターが SBD ノードフェンシングメカニズムを使用できるかどうかを決定するブールフラグ。この変数のデフォルト値は **false** です。
SBD を有効にする **ha_cluster** システムロール Playbook の例については、SBD ノードフェンシングを使用した高可用性クラスターの設定 を参照してください。

ha_cluster_sbd_options

(RHEL 9.1 以降) SBD オプションを指定する名前と値のディクショナリーのリスト。サポートされているオプションは次のとおりです。

- **delay-start** - デフォルトは **no**
- **startmode** - デフォルトは **always**
- **timeout-action** - デフォルトは **flush,reboot**
- **watchdog-timeout** - デフォルトは **5**
  これらのオプションの詳細は、**sbd**(8) man ページの **Configuration via environment** セクションを参照してください。

SBD オプションを設定する **ha_cluster** システムロール Playbook の例については、SBD ノードフェンシングを使用した高可用性クラスターの設定を 参照してください。
SBD を使用する場合、オプションで、インベントリー内のノードごとにウォッチドッグと SBD デバイスを設定できます。インベントリーファイルでウォッチドッグおよび SBD デバイスを設定する方法の詳細は、ha_cluster システムロールのインベントリーの指定 を参照してください。

ha_cluster_cluster_properties

Pacemaker クラスター全体の設定のクラスタープロパティーのセットのリスト。クラスタープロパティーのセットは 1 つだけサポートされます。
クラスタープロパティーのセットの構造は次のとおりです。

```
ha_cluster_cluster_properties:
  - attrs:
      - name: property1_name
        value: property1_value
      - name: property2_name
        value: property2_value
```

デフォルトでは、プロパティーは設定されません。

以下の Playbook の例では、**node1** および **node2** で設定されるクラスターを設定し、**stonith-enabled** および **no-quorum-policy** クラスタープロパティーを設定します。

```
- hosts: node1 node2
  vars:
    ha_cluster_cluster_name: my-new-cluster
    ha_cluster_hacluster_password: password
    ha_cluster_cluster_properties:
      - attrs:
          - name: stonith-enabled
            value: 'true'
          - name: no-quorum-policy
            value: stop

  roles:
    - rhel-system-roles.ha_cluster
```

ha_cluster_resource_primitives

この変数は、stonith リソースなど、システムロールにより設定された Pacemaker リソースを定義します。リソースごとに次の項目を設定できます。

- **id** (必須) - リソースの ID。
- **agent** (必須) - リソースまたは stonith エージェントの名前 (例: **ocf:pacemaker:Dummy** または **stonith:fence_xvm**)。stonith エージェントには、**stonith:** を指定する必要があります。リソースエージェントの場合は、**ocf:pacemaker:Dummy** ではなく、**Dummy** などの短縮名を使用することができます。ただし、同じ名前の複数のエージェントがインストールされていると、使用するエージェントを決定できないため、ロールは失敗します。そのため、リソースエージェントを指定する場合はフルネームを使用することが推奨されます。
- **instance_attrs** (オプション): リソースのインスタンス属性のセットのリスト。現在、1 つのセットのみがサポートされています。属性の名前と値、必須かどうか、およびリソースまたは stonith エージェントによって異なります。
- **meta_attrs** (オプション): リソースのメタ属性のセットのリスト。現在、1 つのセットのみがサポートされています。
- **copy_operations_from_agent** (オプション) - (RHEL 9.3 以降) リソースエージェントは通常、特定のエージェント向けに最適化された、**interval** や **timeout** などのリソース操作のデフォルト設定を定義します。この変数が **true** に設定されている場合、それらの設定がリソース設定にコピーされます。そうでない場合、クラスター全体のデフォルトがリソースに適用されます。**ha_cluster_resource_operation_defaults** ロール変数を使用してリソースのリソース操作のデフォルトも定義する場合は、これを **false** に設定できます。この変数のデフォルト値は **true** です。
- **operations** (任意): リソースの操作のリスト。
  - **action** (必須): pacemaker と、リソースまたは stonith エージェントで定義されている操作アクション。
  - **attrs** (必須): 少なくとも 1 つのオプションを指定する必要があります。

ha_cluster システムロールで設定するリソース定義の構造は次のとおりです。

```
- id: resource-id
  agent: resource-agent
  instance_attrs:
    - attrs:
        - name: attribute1_name
          value: attribute1_value
        - name: attribute2_name
          value: attribute2_value
  meta_attrs:
    - attrs:
        - name: meta_attribute1_name
          value: meta_attribute1_value
        - name: meta_attribute2_name
          value: meta_attribute2_value
  copy_operations_from_agent: bool
  operations:
    - action: operation1-action
      attrs:
        - name: operation1_attribute1_name
          value: operation1_attribute1_value
        - name: operation1_attribute2_name
          value: operation1_attribute2_value
    - action: operation2-action
```

```
      attrs:
        - name: operation2_attribute1_name
          value: operation2_attribute1_value
        - name: operation2_attribute2_name
          value: operation2_attribute2_value
```

デフォルトでは、リソースは定義されません。

リソース設定を含む **ha_cluster** システムロール Playbook の例は、 フェンシングおよびリソースを使用した高可用性クラスターの設定 を参照してください。

ha_cluster_resource_groups

この変数は、システムロールによって設定される Pacemaker リソースグループを定義します。リソースグループごとに次の項目を設定できます。

- **id** (必須): グループの ID。
- **resources** (必須): グループのリソースのリスト。各リソースは ID によって参照され、リソースは **ha_cluster_resource_primitives** 変数に定義する必要があります。1 つ以上のリソースをリスト表示する必要があります。
- **meta_attrs** (オプション): グループのメタ属性のセットのリスト。現在、1 つのセットのみがサポートされています。

ha_cluster システムロールで設定するリソースグループ定義の構造は次のとおりです。

```
ha_cluster_resource_groups:
  - id: group-id
    resource_ids:
      - resource1-id
      - resource2-id
    meta_attrs:
      - attrs:
          - name: group_meta_attribute1_name
            value: group_meta_attribute1_value
          - name: group_meta_attribute2_name
            value: group_meta_attribute2_value
```

デフォルトでは、リソースグループが定義されていません。

リソースグループ設定を含む **ha_cluster** システムロール Playbook の例は、 フェンシングおよびリソースを使用した高可用性クラスターの設定 を参照してください。

ha_cluster_resource_clones

この変数は、システムロールによって設定された Pacemaker リソースクローンを定義します。リソースクローンに対して次の項目を設定できます。

- **resource_id** (必須): クローンを作成するリソース。リソースは **ha_cluster_resource_primitives** 変数または **ha_cluster_resource_groups** 変数に定義する必要があります。
- **promotable** (オプション): 作成するリソースクローンが昇格可能なクローンであるかどうかを示します。これは、**true** または **false** と示されます。

- **id** (任意): クローンのカスタム ID。ID が指定されていない場合は、生成されます。このオプションがクラスターでサポートされない場合は、警告が表示されます。
- **meta_attrs** (任意): クローンのメタ属性のセットのリスト。現在、1つのセットのみがサポートされています。

ha_cluster システムロールで設定するリソースクローン定義の構造は次のとおりです。

```
ha_cluster_resource_clones:
  - resource_id: resource-to-be-cloned
    promotable: true
    id: custom-clone-id
    meta_attrs:
      - attrs:
          - name: clone_meta_attribute1_name
            value: clone_meta_attribute1_value
          - name: clone_meta_attribute2_name
            value: clone_meta_attribute2_value
```

デフォルトでは、リソースのクローンが定義されていません。

リソースクローン設定を含む **ha_cluster** システムロール Playbook の例は、 フェンシングおよびリソースを使用した高可用性クラスターの設定 を参照してください。

ha_cluster_resource_defaults

(RHEL 9.3 以降) この変数は、リソースのデフォルトのセットを定義します。複数のデフォルトのセットを定義し、ルールを使用してそれらを特定のエージェントのリソースに適用できます。**ha_cluster_resource_defaults** 変数で指定したデフォルトは、独自に定義した値で当該デフォルトをオーバーライドするリソースには適用されません。
デフォルトとして指定できるのはメタ属性のみです。

デフォルトセットごとに次の項目を設定できます。

- **id** (オプション) - デフォルトセットの ID。指定しない場合は自動生成されます。
- **rule** (オプション) - いつ、どのリソースにセットを適用するかを定義する **pcs** 構文を使用して記述されたルール。ルールの指定については、**pcs(8)** man ページの **resource defaults set create** セクションを参照してください。
- **score** (オプション) - デフォルトセットの重み。
- **attrs** (オプション) - デフォルトとしてリソースに適用されるメタ属性。

ha_cluster_resource_defaults 変数の構造は次のとおりです。

```
ha_cluster_resource_defaults:
  meta_attrs:
    - id: defaults-set-1-id
      rule: rule-string
      score: score-value
      attrs:
        - name: meta_attribute1_name
          value: meta_attribute1_value
        - name: meta_attribute2_name
          value: meta_attribute2_value
```

```
- id: defaults-set-2-id
  rule: rule-string
  score: score-value
  attrs:
    - name: meta_attribute3_name
      value: meta_attribute3_value
    - name: meta_attribute4_name
      value: meta_attribute4_value
```

リソースのデフォルトを設定する **ha_cluster** システムロール Playbook の例については、リソースおよびリソース操作のデフォルトを使用した高可用性クラスターの設定 を参照してください。

ha_cluster_resource_operation_defaults

(RHEL 9.3 以降) この変数は、リソース操作のデフォルトのセットを定義します。複数のデフォルトのセットを定義し、ルールを使用してそれらを特定のエージェントのリソースおよび特定のリソース操作に適用できます。**ha_cluster_resource_operation_defaults** 変数で指定したデフォルトは、独自に定義した値で当該デフォルトをオーバーライドするリソース操作には適用されません。デフォルトでは、**ha_cluster** システムロールは、リソース操作用の独自の値を定義するようにリソースを設定します。これらのデフォルトを **ha_cluster_resource_operations_defaults** 変数でオーバーライドする方法については、**ha_cluster_resource_primitives** の **copy_operations_from_agent** の項目の説明を参照してください。
デフォルトとして指定できるのはメタ属性のみです。

ha_cluster_resource_operations_defaults 変数の構造は、ルールの指定方法を除き、**ha_cluster_resource_defaults** 変数の構造と同じです。セットが適用されるリソース操作を記述するルールの指定については、**pcs(8)** man ページの **resource op defaults set create** セクションを参照してください。

ha_cluster_constraints_location

この変数は、リソースの場所の制約を定義します。リソースの場所の制約は、リソースを実行できるノードを示します。複数のリソースに一致する可能性のあるリソース ID またはパターンで指定されたリソースを指定できます。ノード名またはルールでノードを指定できます。
リソースの場所の制約に対して次の項目を設定できます。

- **resource** (必須) - 制約が適用されるリソースの仕様。
- **node** (必須) - リソースが優先または回避する必要があるノードの名前。
- **id** (オプション) - 制約の ID。指定しない場合、自動生成されます。
- **options** (オプション) - 名前と値のディクショナリーリスト。
  - **score** - 制約の重みを設定します。
    - 正の **score** 値は、リソースがノードでの実行を優先することを意味します。
    - 負の **score** 値は、リソースがノードで実行されないようにする必要があることを意味します。
    - **-INFINITY** の **score** 値は、リソースがノードで実行されないようにする必要があることを意味します。
    - **score** が指定されていない場合、スコア値はデフォルトで **INFINITY** になります。

デフォルトでは、リソースの場所の制約は定義されていません。

リソース ID とノード名を指定するリソースの場所に対する制約の構造は次のとおりです。

```
ha_cluster_constraints_location:
  - resource:
      id: resource-id
    node: node-name
    id: constraint-id
    options:
      - name: score
        value: score-value
      - name: option-name
        value: option-value
```

リソースパターンを指定するリソースの場所の制約に対して設定する項目は、リソース ID を指定するリソースの場所の制約に対して設定する項目と同じです。ただし、リソース仕様そのものは除きます。リソース仕様に指定する項目は次のとおりです。

- **pattern** (必須) - POSIX 拡張正規表現リソース ID が照合されます。

リソースパターンとノード名を指定するリソースの場所に対する制約の構造は次のとおりです。

```
ha_cluster_constraints_location:
  - resource:
      pattern: resource-pattern
    node: node-name
    id: constraint-id
    options:
      - name: score
        value: score-value
      - name: resource-discovery
        value: resource-discovery-value
```

リソース ID とルールを指定するリソースの場所の制約に対して、次の項目を設定できます。

- **resource** (必須) - 制約が適用されるリソースの仕様。
  - **id** (必須) - リソース ID。
  - **role** (オプション) - 制約が制限されるリソースのロール。**Started**、**Unpromoted**、**Promoted**。
- **rule** (必須) - **pcs** 構文を使用して記述された制約ルール。詳細については、**pcs** (8) の man ページで **constraint location** セクションを参照してください。
- 指定するその他の項目は、ルールを指定しないリソース制約と同じ意味を持ちます。

リソース ID とルールを指定するリソースの場所に対する制約の構造は次のとおりです。

```
ha_cluster_constraints_location:
  - resource:
      id: resource-id
      role: resource-role
    rule: rule-string
    id: constraint-id
    options:
```

```
    - name: score
      value: score-value
    - name: resource-discovery
      value: resource-discovery-value
```

リソースパターンとルールを指定するリソースの場所の制約に対して設定する項目は、リソース ID とルールを指定するリソースの場所の制約に対して設定する項目と同じです。ただし、リソース仕様そのものは除きます。リソース仕様に指定する項目は次のとおりです。

- **pattern** (必須) - POSIX 拡張正規表現リソース ID が照合されます。

リソースパターンとルールを指定するリソースの場所に対する制約の構造は次のとおりです。

```
ha_cluster_constraints_location:
  - resource:
      pattern: resource-pattern
      role: resource-role
    rule: rule-string
    id: constraint-id
    options:
      - name: score
        value: score-value
      - name: resource-discovery
        value: resource-discovery-value
```

リソース制約のあるクラスターを作成する **ha_cluster** システムロール Playbook の例は、リソースに制約のある高可用性クラスターの設定 を参照してください。

ha_cluster_constraints_colocation

この変数は、リソースコロケーションの制約を定義します。リソースコロケーションの制約は、あるリソースの場所が別のリソースの場所に依存することを示しています。コロケーション制約には、2 つのリソースに対する単純なコロケーション制約と、複数のリソースに対するセットコロケーション制約の 2 種類があります。
単純なリソースコロケーション制約に対して次の項目を設定できます。

- **resource_follower** (必須) - **resource_leader** に関連して配置する必要があるリソース。
  - **id** (必須) - リソース ID。
  - **role** (オプション) - 制約が制限されるリソースのロール。**Started**、**Unpromoted**、**Promoted**。
- **resource_leader** (必須) - クラスターは、最初にこのリソースを配置する場所を決定してから、**resource_follower** を配置する場所を決定します。
  - **id** (必須) - リソース ID。
  - **role** (オプション) - 制約が制限されるリソースのロール。**Started**、**Unpromoted**、**Promoted**。
- **id** (オプション) - 制約の ID。指定しない場合、自動生成されます。
- **options** (オプション) - 名前と値のディクショナリーリスト。
  - **score** - 制約の重みを設定します。

- 正の **score** 値は、リソースが同じノードで実行される必要があることを示します。
- 負の **score** 値は、リソースが異なるノードで実行される必要があることを示します。
- **+ INFINITY** の **score** 値は、リソースが同じノードで実行される必要があることを示します。
- **-INFINITY** の **score** 値は、リソースが異なるノードで実行される必要があることを示します。
- **score** が指定されていない場合、スコア値はデフォルトで **INFINITY** になります。

デフォルトでは、リソースコロケーション制約は定義されていません。
単純なリソースコロケーション制約の構造は次のとおりです。

```
ha_cluster_constraints_colocation:
  - resource_follower:
      id: resource-id1
      role: resource-role1
    resource_leader:
      id: resource-id2
      role: resource-role2
    id: constraint-id
    options:
      - name: score
        value: score-value
      - name: option-name
        value: option-value
```

リソースセットのコロケーション制約に対して次の項目を設定できます。

- **resource_sets** (必須) - リソースセットのリスト。
  - **resource_ids** (必須) - セット内のリソースのリスト。
  - **options** (オプション) - セット内のリソースが制約によってどのように扱われるかを微調整する名前と値のディクショナリーリスト。
- **id** (オプション) - 単純なコロケーション制約の場合と同じ値。
- **options** (オプション) - 単純なコロケーション制約の場合と同じ値。

リソースセットに対するコロケーション制約の構造は次のとおりです。

```
ha_cluster_constraints_colocation:
  - resource_sets:
      - resource_ids:
          - resource-id1
          - resource-id2
        options:
          - name: option-name
            value: option-value
    id: constraint-id
    options:
      - name: score
```

```
      value: score-value
    - name: option-name
      value: option-value
```

リソース制約のあるクラスターを作成する **ha_cluster** システムロール Playbook の例は、リソースに制約のある高可用性クラスターの設定 を参照してください。

ha_cluster_constraints_order

この変数は、リソースの順序に対する制約を定義します。リソース順序の制約は、特定のリソースアクションが発生する順序を示します。リソース順序の制約には、2 つのリソースに対する単純な順序制約と、複数のリソースに対するセット順序制約の 2 種類があります。
単純なリソース順序制約に対して次の項目を設定できます。

- **resource_first** (必須) - **resource_then** リソースが依存するリソース。
  - **id** (必須) - リソース ID。
  - **action** (オプション) - **resource_then** リソースに対してアクションを開始する前に完了する必要のあるアクション。許可される値: **start**、**stop**、**promote**、**demode**。
- **resource_then** (必須) - 依存リソース。
  - **id** (必須) - リソース ID。
  - **action** (オプション) - **resource_first** リソースに対するアクションが完了した後にのみリソースが実行できるアクション。許可される値: **start**、**stop**、**promote**、**demode**。
- **id** (オプション) - 制約の ID。指定しない場合、自動生成されます。
- **options** (オプション) - 名前と値のディクショナリーリスト。

デフォルトでは、リソース順序の制約は定義されていません。
単純なリソース順序の制約の構造は次のとおりです。

```
ha_cluster_constraints_order:
  - resource_first:
      id: resource-id1
      action: resource-action1
    resource_then:
      id: resource-id2
      action: resource-action2
    id: constraint-id
    options:
      - name: score
        value: score-value
      - name: option-name
        value: option-value
```

リソースセットの順序制約に対して次の項目を設定できます。

- **resource_sets** (必須) - リソースセットのリスト。
  - **resource_ids** (必須) - セット内のリソースのリスト。
  - **options** (オプション) - セット内のリソースが制約によってどのように扱われるかを微調整する名前と値のディクショナリーリスト。

- **id** (オプション) - 単純な順序制約の場合と同じ値。
- **options** (オプション) - 単純な順序制約の場合と同じ値。

リソースセットの順序制約の構造は次のとおりです。

```
ha_cluster_constraints_order:
  - resource_sets:
      - resource_ids:
          - resource-id1
          - resource-id2
        options:
          - name: option-name
            value: option-value
    id: constraint-id
    options:
      - name: score
        value: score-value
      - name: option-name
        value: option-value
```

リソース制約のあるクラスターを作成する **ha_cluster** システムロール Playbook の例は、リソースに制約のある高可用性クラスターの設定 を参照してください。

ha_cluster_constraints_ticket

この変数は、リソースチケットの制約を定義します。リソースチケットの制約は、特定のチケットに依存するリソースを示します。リソースチケット制約には、1 つのリソースに対する単純なチケット制約と、複数のリソースに対するチケット順序の制約の 2 種類があります。
単純なリソースチケット制約に対して次の項目を設定できます。

- **resource** (必須) - 制約が適用されるリソースの仕様。
  - **id** (必須) - リソース ID。
  - **role** (オプション) - 制約が制限されるリソースのロール。**Started**、**Unpromoted**、**Promoted**。
- **ticket** (必須) - リソースが依存するチケットの名前。
- **id** (オプション) - 制約の ID。指定しない場合、自動生成されます。
- **options** (オプション) - 名前と値のディクショナリーリスト。
  - **loss-policy** (オプション) - チケットが取り消された場合にリソースに対して実行するアクション。

デフォルトでは、リソースチケットの制約は定義されていません。
単純なリソースチケット制約の構造は次のとおりです。

```
ha_cluster_constraints_ticket:
  - resource:
      id: resource-id
      role: resource-role
    ticket: ticket-name
    id: constraint-id
```

```
    options:
      - name: loss-policy
        value: loss-policy-value
      - name: option-name
        value: option-value
```

リソースセットチケット制約に対して次の項目を設定できます。

- **resource_sets** (必須) - リソースセットのリスト。
  - **resource_ids** (必須) - セット内のリソースのリスト。
  - **options** (オプション) - セット内のリソースが制約によってどのように扱われるかを微調整する名前と値のディクショナリーリスト。
- **ticket** (必須) - 単純なチケット制約の場合と同じ値。
- **id** (オプション) - 単純なチケット制約の場合と同じ値。
- **options** (オプション) - 単純なチケット制約の場合と同じ値。

リソースセットのチケット制約の構造は次のとおりです。

```
ha_cluster_constraints_ticket:
  - resource_sets:
      - resource_ids:
          - resource-id1
          - resource-id2
        options:
          - name: option-name
            value: option-value
    ticket: ticket-name
    id: constraint-id
    options:
      - name: option-name
        value: option-value
```

リソース制約のあるクラスターを作成する **ha_cluster** システムロール Playbook の例は、リソースに制約のある高可用性クラスターの設定 を参照してください。

ha_cluster_qnetd

(RHEL 9.1 以降) この変数は、クラスターの外部クォーラムデバイスとして機能できる **qnetd** ホストを設定します。
qnetd ホストに対して次の項目を設定できます。

- **present** (オプション) - **true** の場合、ホスト上に **qnetd** インスタンスを設定します。**false** の場合、ホストから **qnetd** 設定を削除します。デフォルト値は **false** です。これを **true** に設定する場合は、**ha_cluster_cluster_present** を **false** に設定する必要があります。
- **start_on_boot** (オプション) - ブート時に **qnetd** インスタンスを自動的に開始するかどうかを設定します。デフォルト値は **true** です。
- **regenerate_keys** (オプション) - **qnetd** TLS 証明書を再生成するには、この変数を **true** に設定します。証明書を再生成する場合は、各クラスターのロールを再実行して **qnetd** ホストに再度接続するか、手動で **pcs** を実行する必要があります。

フェンシングにより **qnetd** 操作が中断されるため、クラスターノード上で **qnetd** を実行することはできません。
クォーラムデバイスを使用してクラスターを設定する **ha_cluster** システムロール Playbook の例については、クォーラムデバイスを使用したクラスターの設定 を参照してください。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/ha_cluster/** ディレクトリー

25.2. HA_CLUSTER システムロールのインベントリーの指定

ha_cluster システムロール Playbook を使用して HA クラスターを設定する場合は、インベントリー内のクラスターの名前とアドレスを設定します。

25.2.1. インベントリーでのノード名とアドレスの設定

インベントリー内の各ノードには、必要に応じて以下の項目を指定することができます。

- **node_name** - クラスター内のノードの名前。
- **pcs_address** - ノードと通信するために **pcs** が使用するアドレス。名前、FQDN、または IP アドレスを指定でき、ポート番号を含めることができます。
- **corosync_addresses**: Corosync が使用するアドレスのリスト。特定のクラスターを形成するすべてのノードは、同じ数のアドレスが必要で、アドレスの順序が重要です。

以下の例は、ターゲット **node1** および **node2** を持つインベントリーを示しています。**node1** および **node2** は完全修飾ドメイン名のいずれかである必要があります。そうでないと、たとえば、名前が **/etc/hosts** ファイルで解決可能である場合などに、ノードに接続できる必要があります。

```
all:
  hosts:
    node1:
      ha_cluster:
        node_name: node-A
        pcs_address: node1-address
        corosync_addresses:
          - 192.168.1.11
          - 192.168.2.11
    node2:
      ha_cluster:
        node_name: node-B
        pcs_address: node2-address:2224
        corosync_addresses:
          - 192.168.1.12
          - 192.168.2.12
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** ファイル

- **/usr/share/doc/rhel-system-roles/ha_cluster/** ディレクトリー

25.2.2. インベントリーでのウォッチドッグおよび SBD デバイスの設定

(RHEL 9.1 以降) SBD を使用する場合、必要に応じて、インベントリー内のノードごとにウォッチドッグと SBD デバイスを設定できます。すべての SBD デバイスをすべてのノードに共有してアクセスできるようにする必要がありますが、各ノードは各デバイスに異なる名前を使用できます。ウォッチドッグデバイスもノードごとに異なる場合があります。システムロール Playbook で設定できる SBD 変数の詳細は、**ha_cluster** システムロールの変数 の **ha_cluster_sbd_enabled** および **ha_cluster_sbd_options** のエントリーを参照してください。

インベントリー内の各ノードには、必要に応じて以下の項目を指定することができます。

- **sbd_watchdog_modules** (オプション) - (RHEL 9.3 以降) **/dev/watchdog*** デバイスを作成するためにロードするウォッチドッグカーネルモジュール。設定されていない場合、デフォルトで空のリストになります。
- **sbd_watchdog_modules_blocklist** (オプション) - (RHEL 9.3 以降) アンロードおよびブロックするウォッチドッグカーネルモジュール。設定されていない場合、デフォルトで空のリストになります。
- **sbd_watchdog** - SBD が使用するウォッチドッグデバイス。設定されていない場合、デフォルトは **/dev/watchdog** です。
- **sbd_devices** - SBD メッセージの交換と監視に使用するデバイス。設定されていない場合、デフォルトで空のリストになります。

次の例は、ターゲット **node1** および **node2** のウォッチドッグおよび SBD デバイスを設定するインベントリーを示しています。

```
all:
  hosts:
    node1:
      ha_cluster:
        sbd_watchdog_modules:
          - module1
          - module2
        sbd_watchdog: /dev/watchdog2
        sbd_devices:
          - /dev/vdx
          - /dev/vdy
    node2:
      ha_cluster:
        sbd_watchdog_modules:
          - module1
        sbd_watchdog_modules_blocklist:
          - module2
        sbd_watchdog: /dev/watchdog1
        sbd_devices:
          - /dev/vdw
          - /dev/vdz
```

SBD フェンシングを使用する高可用性クラスターの作成については、SBD ノードフェンシングを使用した高可用性クラスターの設定 を参照してください。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/ha_cluster/** ディレクトリー

25.3. 高可用性クラスター用の PCSD TLS 証明書とキーファイルの作成 (RHEL 9.2 以降)

ha_cluster システムロールを使用して、高可用性クラスターに TLS 証明書とキーファイルを作成できます。この Playbook を実行すると、**ha_cluster** システムロールが **certificate** システムロールを内部的に使用して TLS 証明書を管理します。

警告

ha_cluster システムロールは、指定されたノードの既存のクラスター設定を置き換えます。ロールで指定されていない設定は失われます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- クラスターメンバーとして使用するシステムには、RHEL および RHEL High Availability Add-On のアクティブなサブスクリプションがある。
- ha_cluster システムロールのインベントリーの指定 で説明されているように、インベントリーファイルでクラスターノードが指定されている。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Create TLS certificates and key files in a high availability cluster
  hosts: node1 node2
  roles:
    - rhel-system-roles.ha_cluster
  vars:
    ha_cluster_cluster_name: my-new-cluster
    ha_cluster_hacluster_password: <password>
    ha_cluster_manage_firewall: true
    ha_cluster_manage_selinux: true
    ha_cluster_pcsd_certificates:
```

```
    - name: FILENAME
      common_name: "{{ ansible_hostname }}"
      ca: self-sign
```

この Playbook は、**firewalld** および **selinux** サービスを実行するクラスターを設定し、**/var/lib/pcsd** に自己署名の **pcsd** 証明書と秘密鍵ファイルを作成します。**pcsd** 証明書のファイル名は **FILENAME.crt** で、キーファイルの名前は **FILENAME.key** です。

実稼働用の Playbook ファイルを作成するときは、Encrypting content with Ansible Vault で説明されているように、パスワードを vault で暗号化します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/ha_cluster/** ディレクトリー RHEL システムロールを使用した証明書の要求

25.4. リソースを実行していない高可用性クラスターの設定

以下の手順では、**ha_cluster** システムロールを使用して、フェンシングが設定されていない高可用性クラスターと、リソースを実行しない高可用性クラスターを作成します。

警告

ha_cluster システムロールは、指定されたノードの既存のクラスター設定を置き換えます。ロールで指定されていない設定は失われます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- クラスターメンバーとして使用するシステムには、RHEL および RHEL High Availability Add-On のアクティブなサブスクリプションがある。

- ha_cluster システムロールのインベントリーの指定 で説明されているように、インベントリーファイルでクラスターノードが指定されている。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Create a high availability cluster with no fencing and which runs no resources
  hosts: node1 node2
  roles:
    - rhel-system-roles.ha_cluster
  vars:
    ha_cluster_cluster_name: my-new-cluster
    ha_cluster_hacluster_password: <password>
    ha_cluster_manage_firewall: true
    ha_cluster_manage_selinux: true
```

このサンプル Playbook ファイルは、フェンシングを使用せず、リソースを実行しない **firewalld** および **selinux** サービスを実行するクラスターを設定します。

実稼働用の Playbook ファイルを作成するときは、Encrypting content with Ansible Vault で説明されているように、パスワードを vault で暗号化します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/ha_cluster/** ディレクトリー

25.5. フェンシングおよびリソースを使用した高可用性クラスターの設定

以下の手順では、**ha_cluster** システムロールを使用して、フェンスデバイス、クラスターリソース、リソースグループ、およびクローンされたリソースを含む高可用性クラスターを作成します。

警告

ha_cluster システムロールは、指定されたノードの既存のクラスター設定を置き換えます。ロールで指定されていない設定は失われます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- クラスターメンバーとして使用するシステムには、RHEL および RHEL High Availability Add-On のアクティブなサブスクリプションがある。
- ha_cluster システムロールのインベントリーの指定 で説明されているように、インベントリーファイルでクラスターノードが指定されている。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Create a high availability cluster that includes a fencing device and resources
  hosts: node1 node2
  roles:
    - rhel-system-roles.ha_cluster
  vars:
    ha_cluster_cluster_name: my-new-cluster
    ha_cluster_hacluster_password: <password>
    ha_cluster_manage_firewall: true
    ha_cluster_manage_selinux: true
    ha_cluster_resource_primitives:
      - id: xvm-fencing
        agent: 'stonith:fence_xvm'
        instance_attrs:
          - attrs:
              - name: pcmk_host_list
                value: node1 node2
      - id: simple-resource
        agent: 'ocf:pacemaker:Dummy'
      - id: resource-with-options
        agent: 'ocf:pacemaker:Dummy'
        instance_attrs:
          - attrs:
              - name: fake
                value: fake-value
              - name: passwd
                value: passwd-value
        meta_attrs:
          - attrs:
              - name: target-role
                value: Started
              - name: is-managed
                value: 'true'
        operations:
          - action: start
            attrs:
              - name: timeout
```

```
            value: '30s'
        - action: monitor
          attrs:
            - name: timeout
              value: '5'
            - name: interval
              value: '1min'
    - id: dummy-1
      agent: 'ocf:pacemaker:Dummy'
    - id: dummy-2
      agent: 'ocf:pacemaker:Dummy'
    - id: dummy-3
      agent: 'ocf:pacemaker:Dummy'
    - id: simple-clone
      agent: 'ocf:pacemaker:Dummy'
    - id: clone-with-options
      agent: 'ocf:pacemaker:Dummy'
  ha_cluster_resource_groups:
    - id: simple-group
      resource_ids:
        - dummy-1
        - dummy-2
      meta_attrs:
        - attrs:
            - name: target-role
              value: Started
            - name: is-managed
              value: 'true'
    - id: cloned-group
      resource_ids:
        - dummy-3
  ha_cluster_resource_clones:
    - resource_id: simple-clone
    - resource_id: clone-with-options
      promotable: yes
      id: custom-clone-id
      meta_attrs:
        - attrs:
            - name: clone-max
              value: '2'
            - name: clone-node-max
              value: '1'
    - resource_id: cloned-group
      promotable: yes
```

このサンプル Playbook ファイルは、**firewalld** および **selinux** サービスを実行するクラスターを設定します。クラスターには、フェンシング、いくつかのリソース、およびリソースグループが含まれています。また、リソースグループのリソースクローンも含まれます。

実稼働用の Playbook ファイルを作成するときは、Encrypting content with Ansible Vault で説明されているように、パスワードを vault で暗号化します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/ha_cluster/** ディレクトリー

25.6. リソースおよびリソース操作のデフォルトを使用した高可用性クラスターの設定

(RHEL 9.3 以降) 次の手順では、**ha_cluster** システムロールを使用して、リソースとリソース操作のデフォルトを定義する高可用性クラスターを作成します。

警告

ha_cluster システムロールは、指定されたノードの既存のクラスター設定を置き換えます。ロールで指定されていない設定は失われます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- クラスターメンバーとして使用するシステムには、RHEL および RHEL High Availability Add-On のアクティブなサブスクリプションがある。
- ha_cluster システムロールのインベントリーの指定 で説明されているように、インベントリーファイルでクラスターノードが指定されている。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- name: Create a high availability cluster that defines resource and resource operation defaults
  hosts: node1 node2
  roles:
    - rhel-system-roles.ha_cluster
  vars:
```

```
    ha_cluster_cluster_name: my-new-cluster
    ha_cluster_hacluster_password: <password>
    # Set a different resource-stickiness value during
    # and outside work hours. This allows resources to
    # automatically move back to their most
    # preferred hosts, but at a time that
    # does not interfere with business activities.
    ha_cluster_resource_defaults:
      meta_attrs:
        - id: core-hours
          rule: date-spec hours=9-16 weekdays=1-5
          score: 2
          attrs:
            - name: resource-stickiness
              value: INFINITY
        - id: after-hours
          score: 1
          attrs:
            - name: resource-stickiness
              value: 0
    # Default the timeout on all 10-second-interval
    # monitor actions on IPaddr2 resources to 8 seconds.
    ha_cluster_resource_operation_defaults:
      meta_attrs:
        - rule: resource ::IPaddr2 and op monitor interval=10s
          score: INFINITY
          attrs:
            - name: timeout
              value: 8s
```

このサンプル Playbook ファイルは、**firewalld** および **selinux** サービスを実行するクラスターを設定します。クラスターには、リソースとリソース操作のデフォルトが含まれます。

実稼働用の Playbook ファイルを作成するときは、Encrypting content with Ansible Vault で説明されているように、パスワードを vault で暗号化します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/ha_cluster/** ディレクトリー

25.7. リソースに制約のある高可用性クラスターの設定

次の手順では、**ha_cluster** システムロールを使用して、リソースの場所の制約、リソースコロケーションの制約、リソース順序の制約、およびリソースチケットの制約を含む高可用性クラスターを作成します。

警告

ha_cluster システムロールは、指定されたノードの既存のクラスター設定を置き換えます。ロールで指定されていない設定は失われます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- クラスターメンバーとして使用するシステムには、RHEL および RHEL High Availability Add-On のアクティブなサブスクリプションがある。
- ha_cluster システムロールのインベントリーの指定 で説明されているように、インベントリーファイルでクラスターノードが指定されている。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Create a high availability cluster with resource constraints
  hosts: node1 node2
  roles:
    - rhel-system-roles.ha_cluster
  vars:
    ha_cluster_cluster_name: my-new-cluster
    ha_cluster_hacluster_password: <password>
    ha_cluster_manage_firewall: true
    ha_cluster_manage_selinux: true
    # In order to use constraints, we need resources the constraints will apply
    # to.
    ha_cluster_resource_primitives:
      - id: xvm-fencing
        agent: 'stonith:fence_xvm'
        instance_attrs:
          - attrs:
              - name: pcmk_host_list
                value: node1 node2
      - id: dummy-1
        agent: 'ocf:pacemaker:Dummy'
      - id: dummy-2
        agent: 'ocf:pacemaker:Dummy'
```

```
    - id: dummy-3
      agent: 'ocf:pacemaker:Dummy'
    - id: dummy-4
      agent: 'ocf:pacemaker:Dummy'
    - id: dummy-5
      agent: 'ocf:pacemaker:Dummy'
    - id: dummy-6
      agent: 'ocf:pacemaker:Dummy'
  # location constraints
  ha_cluster_constraints_location:
    # resource ID and node name
    - resource:
        id: dummy-1
      node: node1
      options:
        - name: score
          value: 20
    # resource pattern and node name
    - resource:
        pattern: dummy-\d+
      node: node1
      options:
        - name: score
          value: 10
    # resource ID and rule
    - resource:
        id: dummy-2
      rule: '#uname eq node2 and date in_range 2022-01-01 to 2022-02-28'
    # resource pattern and rule
    - resource:
        pattern: dummy-\d+
      rule: node-type eq weekend and date-spec weekdays=6-7
  # colocation constraints
  ha_cluster_constraints_colocation:
    # simple constraint
    - resource_leader:
        id: dummy-3
      resource_follower:
        id: dummy-4
      options:
        - name: score
          value: -5
    # set constraint
    - resource_sets:
        - resource_ids:
            - dummy-1
            - dummy-2
        - resource_ids:
            - dummy-5
            - dummy-6
          options:
            - name: sequential
              value: "false"
      options:
        - name: score
          value: 20
```

```
    # order constraints
ha_cluster_constraints_order:
  # simple constraint
  - resource_first:
      id: dummy-1
    resource_then:
      id: dummy-6
    options:
      - name: symmetrical
        value: "false"
  # set constraint
  - resource_sets:
      - resource_ids:
          - dummy-1
          - dummy-2
        options:
          - name: require-all
            value: "false"
          - name: sequential
            value: "false"
      - resource_ids:
          - dummy-3
      - resource_ids:
          - dummy-4
          - dummy-5
        options:
          - name: sequential
            value: "false"
# ticket constraints
ha_cluster_constraints_ticket:
  # simple constraint
  - resource:
      id: dummy-1
    ticket: ticket1
    options:
      - name: loss-policy
        value: stop
  # set constraint
  - resource_sets:
      - resource_ids:
          - dummy-3
          - dummy-4
          - dummy-5
    ticket: ticket2
    options:
      - name: loss-policy
        value: fence
```

このサンプル Playbook ファイルは、**firewalld** および **selinux** サービスを実行するクラスターを設定します。クラスターには、リソースの場所の制約、リソースのコロケーションの制約、リソースの順序の制約、およびリソースチケットの制約が含まれます。

実稼働用の Playbook ファイルを作成するときは、Encrypting content with Ansible Vault で説明されているように、パスワードを vault で暗号化します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/ha_cluster/** ディレクトリー

25.8. 高可用性クラスターでの COROSYNC 値の設定

(RHEL 9.1 以降) 次の手順では、**ha_cluster** システムロールを使用して、Corosync 値を設定する高可用性クラスターを作成します。

警告

ha_cluster システムロールは、指定されたノードの既存のクラスター設定を置き換えます。ロールで指定されていない設定は失われます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- クラスターメンバーとして使用するシステムには、RHEL および RHEL High Availability Add-On のアクティブなサブスクリプションがある。
- ha_cluster システムロールのインベントリーの指定 で説明されているように、インベントリーファイルでクラスターノードが指定されている。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- name: Create a high availability cluster that configures Corosync values
  hosts: node1 node2
  roles:
    - rhel-system-roles.ha_cluster
```

```
vars:
  ha_cluster_cluster_name: my-new-cluster
  ha_cluster_hacluster_password: <password>
  ha_cluster_manage_firewall: true
  ha_cluster_manage_selinux: true
  ha_cluster_transport:
    type: knet
    options:
      - name: ip_version
        value: ipv4-6
      - name: link_mode
        value: active
    links:
      -
        - name: linknumber
          value: 1
        - name: link_priority
          value: 5
      -
        - name: linknumber
          value: 0
        - name: link_priority
          value: 10
    compression:
      - name: level
        value: 5
      - name: model
        value: zlib
    crypto:
      - name: cipher
        value: none
      - name: hash
        value: none
  ha_cluster_totem:
    options:
      - name: block_unlisted_ips
        value: 'yes'
      - name: send_join
        value: 0
  ha_cluster_quorum:
    options:
      - name: auto_tie_breaker
        value: 1
      - name: wait_for_all
        value: 1
```

このサンプル Playbook ファイルは、Corosync プロパティーを設定する **firewalld** および **selinux** サービスを実行するクラスターを設定します。

実稼働用の Playbook ファイルを作成するときは、Encrypting content with Ansible Vault で説明されているように、パスワードを vault で暗号化します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/ha_cluster/** ディレクトリー

25.9. SBD ノードフェンシングを使用した高可用性クラスターの設定

(RHEL 9.1 以降) 次の手順では、**ha_cluster** システムロールを使用して、SBD ノードフェンシングを使用する高可用性クラスターを作成します。

警告

ha_cluster システムロールは、指定されたノードの既存のクラスター設定を置き換えます。ロールで指定されていない設定は失われます。

この Playbook は インベントリーでのウォッチドッグおよび SBD デバイスの設定 で説明されているように、ウォッチドッグモジュール (RHEL 9.3 以降でサポート) をロードするインベントリーファイルを使用します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- クラスターメンバーとして使用するシステムには、RHEL および RHEL High Availability Add-On のアクティブなサブスクリプションがある。
- ha_cluster システムロールのインベントリーの指定 で説明されているように、インベントリーファイルでクラスターノードが指定されている。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Create a high availability cluster that uses SBD node fencing
  hosts: node1 node2
  roles:
```

```
    - rhel-system-roles.ha_cluster
  vars:
    ha_cluster_cluster_name: my-new-cluster
    ha_cluster_hacluster_password: <password>
    ha_cluster_manage_firewall: true
    ha_cluster_manage_selinux: true
    ha_cluster_sbd_enabled: yes
    ha_cluster_sbd_options:
      - name: delay-start
        value: 'no'
      - name: startmode
        value: always
      - name: timeout-action
        value: 'flush,reboot'
      - name: watchdog-timeout
        value: 30
    # Suggested optimal values for SBD timeouts:
    # watchdog-timeout * 2 = msgwait-timeout (set automatically)
    # msgwait-timeout * 1.2 = stonith-timeout
    ha_cluster_cluster_properties:
      - attrs:
          - name: stonith-timeout
            value: 72
    ha_cluster_resource_primitives:
      - id: fence_sbd
        agent: 'stonith:fence_sbd'
        instance_attrs:
          - attrs:
              # taken from host_vars
              - name: devices
                value: "{{ ha_cluster.sbd_devices | join(',') }}"
              - name: pcmk_delay_base
                value: 30
```

このサンプル Playbook ファイルは、SBD フェンシングを使用し、SBD Stonith リソースを作成する**firewalld** および **selinux** サービスを実行するクラスターを設定します。

実稼働用の Playbook ファイルを作成するときは、Encrypting content with Ansible Vault で説明されているように、パスワードを vault で暗号化します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** ファイル

- **/usr/share/doc/rhel-system-roles/ha_cluster/** ディレクトリー

25.10. クォーラムデバイスを使用する高可用性クラスターの設定 (RHEL 9.2 以降)

ha_cluster システムロールを使用し、別個のクォーラムデバイスを使用した高可用性クラスターを設定するには、まずクォーラムデバイスをセットアップします。クォーラムデバイスをセットアップした後は、任意の数のクラスターでデバイスを使用できます。

25.10.1. クォーラムデバイスの設定

ha_cluster システムロールを使用してクォーラムデバイスを設定するには、次の手順に従います。クラスターノード上ではクォーラムデバイスを実行できないことに注意してください。

警告

ha_cluster システムロールは、指定されたノードの既存のクラスター設定を置き換えます。ロールで指定されていない設定は失われます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- クォーラムデバイスの実行に使用するシステムには、RHEL および RHEL High Availability Add-On のアクティブなサブスクリプションがある。
- ha_cluster システムロールのインベントリーの指定 で説明されているように、インベントリーファイルでクォーラムデバイスが指定されている。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Configure a quorum device
  hosts: nodeQ
  roles:
    - rhel-system-roles.ha_cluster
  vars:
    ha_cluster_cluster_present: false
    ha_cluster_hacluster_password: <password>
    ha_cluster_manage_firewall: true
    ha_cluster_manage_selinux: true
    ha_cluster_qnetd:
      present: true
```

このサンプル Playbook ファイルは、**firewalld** および **selinux** サービスを実行しているシステムにクォーラムデバイスを設定します。

実稼働用の Playbook ファイルを作成するときは、Encrypting content with Ansible Vault で説明されているように、パスワードを vault で暗号化します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/ha_cluster/** ディレクトリー

25.10.2. クォーラムデバイスを使用するようにクラスターを設定する

クォーラムデバイスを使用するようにクラスターを設定するには、次の手順に従います。

警告

ha_cluster システムロールは、指定されたノードの既存のクラスター設定を置き換えます。ロールで指定されていない設定は失われます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- クラスターメンバーとして使用するシステムには、RHEL および RHEL High Availability Add-On のアクティブなサブスクリプションがある。
- ha_cluster システムロールのインベントリーの指定 で説明されているように、インベントリーファイルでクラスターノードが指定されている。
- クォーラムデバイスが設定されている。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Configure a cluster to use a quorum device
  hosts: node1 node2
  roles:
    - rhel-system-roles.ha_cluster
  vars:
    ha_cluster_cluster_name: my-new-cluster
    ha_cluster_hacluster_password: <password>
    ha_cluster_manage_firewall: true
    ha_cluster_manage_selinux: true
    ha_cluster_quorum:
      device:
        model: net
        model_options:
          - name: host
            value: nodeQ
          - name: algorithm
            value: lms
```

このサンプル Playbook ファイルは、クォーラムデバイスを使用する **firewalld** および **selinux** サービスを実行するクラスターを設定します。

実稼働用の Playbook ファイルを作成するときは、Encrypting content with Ansible Vault で説明されているように、パスワードを vault で暗号化します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/ha_cluster/** ディレクトリー

25.11. HA_CLUSTER システムロールを使用した高可用性クラスターでの APACHE HTTP サーバーの設定

この手順では、**ha_cluster** システムロールを使用して、2 ノードの Red Hat Enterprise Linux High Availability Add-On クラスターでアクティブ/パッシブな Apache HTTP サーバーを設定します。

警告

ha_cluster システムロールは、指定されたノードの既存のクラスター設定を置き換えます。ロールで指定されていない設定は失われます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- クラスターメンバーとして使用するシステムには、RHEL および RHEL High Availability Add-On のアクティブなサブスクリプションがある。
- ha_cluster システムロールのインベントリーの指定 で説明されているように、インベントリーファイルでクラスターノードが指定されている。
- Pacemaker クラスターで XFS ファイルシステムを持つ LVM ボリュームを設定する の説明に従って、XFS ファイルシステムを使用して LVM 論理ボリュームを設定している。
- Configuring an Apache HTTP Server の説明に従って、Apache HTTP サーバーを設定している。
- システムには、クラスターノードをフェンスするのに使用される APC 電源スイッチが含まれている。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Configure active/passive Apache server in a high availability cluster
  hosts: z1.example.com z2.example.com
  roles:
    - rhel-system-roles.ha_cluster
  vars:
    ha_cluster_hacluster_password: <password>
    ha_cluster_cluster_name: my_cluster
    ha_cluster_manage_firewall: true
    ha_cluster_manage_selinux: true
    ha_cluster_fence_agent_packages:
      - fence-agents-apc-snmp
    ha_cluster_resource_primitives:
      - id: myapc
        agent: stonith:fence_apc_snmp
        instance_attrs:
          - attrs:
              - name: ipaddr
                value: zapc.example.com
```

```
            - name: pcmk_host_map
              value: z1.example.com:1;z2.example.com:2
            - name: login
              value: apc
            - name: passwd
              value: apc
      - id: my_lvm
        agent: ocf:heartbeat:LVM-activate
        instance_attrs:
          - attrs:
            - name: vgname
              value: my_vg
            - name: vg_access_mode
              value: system_id
      - id: my_fs
        agent: Filesystem
        instance_attrs:
          - attrs:
            - name: device
              value: /dev/my_vg/my_lv
            - name: directory
              value: /var/www
            - name: fstype
              value: xfs
      - id: VirtualIP
        agent: IPaddr2
        instance_attrs:
          - attrs:
            - name: ip
              value: 198.51.100.3
            - name: cidr_netmask
              value: 24
      - id: Website
        agent: apache
        instance_attrs:
          - attrs:
            - name: configfile
              value: /etc/httpd/conf/httpd.conf
            - name: statusurl
              value: http://127.0.0.1/server-status
    ha_cluster_resource_groups:
      - id: apachegroup
        resource_ids:
          - my_lvm
          - my_fs
          - VirtualIP
          - Website
```

このサンプル Playbook ファイルは、**firewalld** および **selinux** サービスを実行するアクティブ/パッシブ 2 ノード HA クラスターに、以前に作成した Apache HTTP サーバーを設定します。

この例では、ホスト名が **zapc.example.com** の APC 電源スイッチを使用します。クラスターが他のフェンスエージェントを使用しない場合は、以下の例のように、**ha_cluster_fence_agent_packages** 変数を定義するときに、クラスターが必要とするフェンスエージェントのみを任意でリスト表示できます。

実稼働用の Playbook ファイルを作成するときは、Encrypting content with Ansible Vault で説明されているように、パスワードを vault で暗号化します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

4. **apache** リソースエージェントを使用して Apache を管理する場合は **systemd** が使用されません。このため、Apache で提供される **logrotate** スクリプトを編集して、**systemctl** を使用して Apache を再ロードしないようにする必要があります。
クラスター内の各ノードで、**/etc/logrotate.d/httpd** ファイルから以下の行を削除します。

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

削除した行を次の 3 行に置き換え、PID ファイルパスとして **/var/run/httpd-website.pid** を指定します。ここの **website** は、Apache リソースの名前になります。この例では、Apache リソース名は **Website** です。

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

検証

1. クラスター内のノードのいずれかから、クラスターのステータスを確認します。4 つのリソースがすべて同じノード (**z1.example.com**) で実行されていることに注意してください。
設定したリソースが実行していない場合は、**pcs resource debug-start resource** コマンドを実行して、リソースの設定をテストします。

```
[root@z1 ~]# pcs status
Cluster name: my_cluster
Last updated: Wed Jul 31 16:38:51 2013
Last change: Wed Jul 31 16:42:14 2013 via crm_attribute on z1.example.com
Stack: corosync
Current DC: z2.example.com (2) - partition with quorum
Version: 1.1.10-5.el7-9abe687
2 Nodes configured
6 Resources configured

Online: [ z1.example.com z2.example.com ]

Full list of resources:
 myapc  (stonith:fence_apc_snmp):       Started z1.example.com
 Resource Group: apachegroup
     my_lvm     (ocf::heartbeat:LVM-activate):   Started z1.example.com
```

```
    my_fs      (ocf::heartbeat:Filesystem):    Started z1.example.com
    VirtualIP  (ocf::heartbeat:IPaddr2):       Started z1.example.com
    Website    (ocf::heartbeat:apache):        Started z1.example.com
```

2. クラスターが稼働したら、ブラウザーで、**IPaddr2** リソースとして定義した IP アドレスを指定して、Hello と単語が表示されるサンプル表示を確認します。

```
Hello
```

3. **z1.example.com** で実行しているリソースグループが **z2.example.com** ノードにフェイルオーバーするかどうかをテストするには、ノード **z1.example.com** を **standby** にすると、ノードがリソースをホストできなくなります。

```
[root@z1 ~]# pcs node standby z1.example.com
```

4. ノード **z1** を **standby** モードにしたら、クラスター内のノードのいずれかからクラスターのステータスを確認します。リソースはすべて **z2** で実行しているはずです。

```
[root@z1 ~]# pcs status
Cluster name: my_cluster
Last updated: Wed Jul 31 17:16:17 2013
Last change: Wed Jul 31 17:18:34 2013 via crm_attribute on z1.example.com
Stack: corosync
Current DC: z2.example.com (2) - partition with quorum
Version: 1.1.10-5.el7-9abe687
2 Nodes configured
6 Resources configured

Node z1.example.com (1): standby
Online: [ z2.example.com ]

Full list of resources:

 myapc  (stonith:fence_apc_snmp):       Started z1.example.com
 Resource Group: apachegroup
    my_lvm     (ocf::heartbeat:LVM-activate):   Started z2.example.com
    my_fs      (ocf::heartbeat:Filesystem):    Started z2.example.com
    VirtualIP  (ocf::heartbeat:IPaddr2):       Started z2.example.com
    Website    (ocf::heartbeat:apache):        Started z2.example.com
```

定義している IP アドレスの Web サイトは、中断せず表示されているはずです。

5. **スタンバイ** モードから **z1** を削除するには、以下のコマンドを実行します。

```
[root@z1 ~]# pcs node unstandby z1.example.com
```

注記

ノードを **スタンバイ** モードから削除しても、リソースはそのノードにフェイルオーバーしません。これは、リソースの **resource-stickiness** 値により異なります。**resource-stickiness** メタ属性については、現在のノードを優先するようにリソースを設定する を参照してください。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.ha_cluster/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/ha_cluster/** ディレクトリー

第26章 COCKPIT RHEL システムロールを使用した WEB コンソールのインストールと設定

cockpit RHEL システムロールを使用すると、システムに Web コンソールをインストールして設定できます。

26.1. COCKPIT システムロール

cockpit システムロールを使用して、Web コンソールを自動的にデプロイして有効にできます。その結果、Web ブラウザーから RHEL システムを管理できるようになります。

26.2. COCKPIT RHEL システムロールの変数

cockpit RHEL システムロールに使用されるパラメーターは次のとおりです。

ロール変数	説明
cockpit_packages: (default: default)	事前定義されたパッケージセット (default、minimal、full) の 1 つを設定します。 * cockpit_packages: (default: default) - 最も一般的なページとオンデマンドインストール UI * cockpit_packages: (default: minimal) - 概要、ターミナル、ログ、アカウント、およびメトリックのページのみ。最小限の依存関係。 * cockpit_packages: (default: full) - 利用可能なすべてのページ。 必要に応じて、インストールするコックピットパッケージを独自に選択します。
cockpit_enabled: (default:true)	Web コンソール Web サーバーが起動時に自動起動できるかどうかを設定します。
cockpit_started: (default:true)	Web コンソールを起動するかどうかを設定します。
cockpit_config: (default: nothing)	**/etc/cockpit/cockpit.conf** ファイルで設定を適用できます。注記: 以前の設定ファイルは失われます。
cockpit_port: (default: 9090)	Web コンソールはデフォルトでポート 9090 で実行されます。このオプションを使用してポートを変更できます。
cockpit_manage_firewall: (default: false)	**cockpit** ロールが **firewall** ロールを制御してポートを追加できるようにします。ポートの削除には使用できません。ポートを削除する場合は、ファイアウォールシステムロールを直接使用する必要があります。

ロール変数	説明
cockpit_manage_selinux: (default: false)	**cockpit** ロールが **selinux** ロールを使用して SELinux を設定できるようにします。デフォルトの SELinux ポリシーでは、Cockpit がポート 9090 以外でリッスンすることは許可されていません。ポートを変更する場合は、**selinux** ロールが正しいポートパーミッション (**websm_port_t**) を設定できるように、このオプションを **true** に設定します。
cockpit_certificates: (default: nothing)	**cockpit** ロールが **certificate** ロールを使用して、新しい証明書を生成できるようにします。**cockpit_certificates** の値は、**certificate** ロールの **certificate_requests** 変数に渡されます。このロールは **cockpit** ロールによって内部的に呼び出され、秘密鍵と証明書を生成します。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.cockpit/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/cockpit/** ディレクトリー
- cockpit.conf(5) man ページ

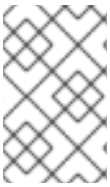
26.3. COCKPIT RHEL システムロールを使用した WEB コンソールのインストール

cockpit システムロールを使用して、RHEL Web コンソールをインストールして有効にすることができます。

デフォルトでは、RHEL Web コンソールは自己署名証明書を使用します。セキュリティー上の理由から、代わりに信頼された認証局によって発行された証明書を指定できます。

この例では、**cockpit** システムロールを使用して次のことを行います。

- RHEL Web コンソールをインストールする
- Web コンソールが **firewalld** を管理できるようにする
- 自己署名証明書を使用する代わりに、**ipa** の信頼された認証局からの証明書を使用するように Web コンソールを設定する
- カスタムポート 9050 を使用するように Web コンソールを設定する



注記

ファイアウォールを管理したり証明書を作成したりするために、Playbook で **firewall** または **certificate** システムロールを呼び出す必要はありません。**cockpit** システムロールが、必要に応じてそれらを自動的に呼び出します。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Manage the RHEL web console
  hosts: managed-node-01.example.com
  tasks:
    - name: Install RHEL web console
      ansible.builtin.include_role:
        name: rhel-system-roles.cockpit
      vars:
        cockpit_packages: default
        cockpit_port:9050
        cockpit_manage_selinux: true
        cockpit_manage_firewall: true
        cockpit_certificates:
          - name: /etc/cockpit/ws-certs.d/01-certificate
            dns: ['localhost', 'www.example.com']
            ca: ipa
            group: cockpit-ws
```

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.cockpit/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/cockpit** ディレクトリー
- RHEL システムロールを使用した証明書の要求

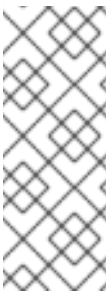
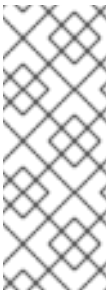
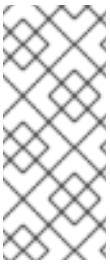
第27章 PODMAN RHEL システムロールを使用したコンテナーの管理

podman RHEL システムロールを使用すると、Podman 設定、コンテナー、および Podman コンテナーを実行する **systemd** サービスを管理できます。

27.1. PODMAN RHEL システムロールの変数

podman RHEL システムロールに使用されるパラメーターは次のとおりです。

変数	説明
podman_kube_specs	Podman Pod および対応する systemd ユニットについて説明します。 ● **state**: (デフォルト:**created**) - **systemd** サービスと Pod で実行される操作を示します。 ○ **created**: Pod と **systemd** サービスを作成しますが、実行しません。 ○ **started**: Pod と **systemd** サービスを作成して起動します。 ○ **absent**: Pod と **systemd** サービスを削除します。 ● **run_as_user**: (デフォルト: **podman_run_as_user**) - Pod ごとのユーザー。ユーザーを指定しない場合は、**root** が使用されます。 **注記** ユーザーはすでに存在している必要があります。 ● **run_as_group** (デフォルト: **podman_run_as_group**) - Pod ごとのグループ。ユーザーを指定しない場合は、**root** が使用されます。 **注記** グループはすでに存在している必要があります。 ● **systemd_unit_scope** (デフォルト: **podman_systemd_unit_scope**) - **systemd** ユニットに使用するスコープ。指定しない場合、ルートコンテナーには **system** が使用され、ユーザーコンテナーには **user** が使用されます。 ● **kube_file_src** - 管理対象ノードの **kube_file** にコピーされるコントローラー

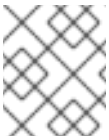
変数	説明
	ノード上の Kubernetes YAML ファイルの名前。  **注記** **kube_file_content** 変数を指定する場合は、**kube_file_src** 変数を指定しないでください。**kube_file_content** は、**kube_file_src** よりも優先されます。 ● **kube_file_content** - Kubernetes YAML 形式の文字列、または Kubernetes YAML 形式の dict。管理対象ノード上の **kube_file** の内容を指定します。  **注記** **kube_file_src** 変数を指定する場合は、**kube_file_content** 変数を指定しないでください。**kube_file_content** は、**kube_file_src** よりも優先されます。 ● **kube_file** - コンテナーまたは Pod の Kubernetes 仕様を含む管理対象ノード上のファイルの名前。通常、**kube_file** ファイルをロール外の管理対象ノードにコピーする必要がない限り、**kube_file** 変数を指定する必要はありません。**kube_file_src** または **kube_file_content** のいずれかを指定する場合は、これを指定する必要はありません。  **注記** **kube_file** を省略し、代わりに **kube_file_src** または **kube_file_content** を指定し、ロールにファイルのパスと名前を管理させることを強く推奨します。 ○ ファイルのベース名は、K8s yaml の metadata.name 値に **.yml** 接尾辞が追加されたものになります。 ○ システムサービスのディレクトリーは **/etc/containers/ansible-kubernetes.d** です。 ○ ユーザーサービスのディレクトリーは **$HOME/.config/containers/ansible-kubernetes.d** です。 ○ これは、管理対象ノード上の **/etc/containers/ansible-**

変数	説明 kubernetes.d/<application_name>.yml ファイルにコピーされます。
podman_quadlet_specs	Quadlet 仕様 のリスト。 **警告** Quadlet は、RHEL 8 ではルートフルコンテナーでのみ機能します。Quadlet は、RHEL 9 でのみルートレスコンテナーで動作します。 Quadlet はユニットの名前とタイプによって定義されます。ユニットのタイプには、**container**、**kube**、**network**、および **volume** があります。**name** と **type** を明示的に渡すことも、**name** と **type** を **file**、**file_src**、または **template_src** で指定されたファイル名から導出することもできます。 ● ルートコンテナーファイルは、管理対象ノードの **/etc/containers/systemd/$name.$type** にあります。 ● ルートレスコンテナーファイルは、管理対象ノードの **$HOME/.config/containers/systemd/$name.$type** にあります。 Quadlet 仕様が他のファイルに依存する場合、たとえば **quadlet.kube** が **Yaml** ファイルまたは **ConfigMap** に依存する場合、依存先のファイルは、**podman_quadlet_specs** リストで、そのファイルを使用するファイルの前に指定する必要があります。たとえば、**my-app.kube** ファイルがあるとします。 [Kube] ConfigMap=my-app-config.yml Yaml=my-app.yml ... その場合、**my-app.kube** の前に **my-app-config.yml** と **my-app.yml** を指定する必要があります。 podman_quadlet_specs: - file_src: my-app-config.yml - file_src: my-app.yml - file_src: my-app.kube

変数	説明
	各 Quadlet 仕様のパラメーターのほとんどは、**kube** パラメーターがサポートされていないことを除き、上記の **podman_kube_spec** のパラメーターと同じです。次のパラメーターがサポートされています。 ● **name** - ユニットの名前。名前を指定しない場合は、**file**、**file_src**、または **template_src** から導出されます。 ○ たとえば、**file_src:/path/to/my-container.container** と指定した場合、**name** は **my-container** になります。 ● **type** - ユニットのタイプには、**container**、**kube**、**network**、**volume** があります。名前を指定しない場合は、**file**、**file_src**、または **template_src** から導出されます。 ○ たとえば、**file_src:/path/to/my-container.container** と指定した場合、**type** は **container** になります。  **注記** このファイルが Quadlet ユニット形式で、有効な Quadlet ユニット接尾辞がある場合は、Quadlet ユニットとして使用されます。それ以外の場合は、単にコピーされます。 ● **file_src** - Quadlet ユニットのソースとして使用するために管理対象ノードにコピーする、コントロールノード上のファイルの名前。  **注記** このファイルが Quadlet ユニット形式で、有効な Quadlet ユニット接尾辞がある場合は、Quadlet ユニットとして使用されます。それ以外の場合は、単にコピーされます。 ● **file** - Quadlet ユニットのソースとして使用する管理対象ノード上のファイルの名前。

変数	説明
	注記 このファイルが Quadlet ユニット形式で、有効な Quadlet ユニット接尾辞がある場合は、Quadlet ユニットとして使用されます。それ以外の場合は、単にコピーされます。 - **file_content** - 管理対象ノードにコピーするファイルの内容 (文字列形式)。これは、インラインで簡単に指定できる小さなファイルを渡すのに便利です。**name** と **type** を指定する必要があります。 - **template_src** - Jinja * **template** ファイルとして処理され、Quadlet ユニットのソースとして使用するために管理対象ノードにコピーされるコントロールノード上のファイルの名前。 **注記** このファイルが Quadlet ユニット形式で、有効な Quadlet ユニット接尾辞がある場合は、Quadlet ユニットとして使用されます。それ以外の場合は、単にコピーされます。ファイルに **.j2** 接尾辞が付いている場合、その接尾辞は quadlet ファイルタイプを決定するために削除されます。 o たとえば、次のように指定するとします。 podman_quadlet_specs: - template_src: my-app.container.j2 この場合、ローカルファイル **templates/my-app.container.j2** が Jinja テンプレートファイルとして処理され、管理対象ノード上の Quadlet コンテナーユニット仕様として **/etc/containers/systemd/my-app.container** にコピーされます。

変数	説明
podman_secrets	podman_secret で使用されるのと同じ形式のシークレット仕様のリスト。ただし、指定されたユーザーのアカウントでシークレットを作成するために使用される追加のフィールド **run_as_user** がある点が異なります。このフィールドが指定されていない場合、シークレットは **podman_run_as_user** で指定されたアカウントに作成されます。デフォルト値は "root" です。Ansible Vault を使用して **data** フィールドの値を暗号化します。
podman_create_host_directories	true の場合、ロールは、**podman_kube_specs** で指定された Kubernetes YAML の **volume.hostPath** 仕様のホストマウントで指定されたホストディレクトリーを確実にします。デフォルト値は false です。  **注記** ロールでディレクトリーを確実に管理するには、ディレクトリーを絶対パス (ルートコンテナーの場合) またはホームディレクトリーに対する相対パス (非ルートコンテナーの場合) として指定する必要があります。 このロールは、デフォルトの所有権またはパーミッションをディレクトリーに適用します。所有権またはパーミッションを設定する必要がある場合は、**podman_host_directories** を参照してください。
podman_host_directories	これは dict です。**podman_create_host_directories** を使用して、ボリュームマウント用のホストディレクトリーを作成するようにロールに指示し、これらの作成されたホストディレクトリーに適用されるパーミッションまたは所有権を指定する必要がある場合は、**podman_host_directories** を使用します。各キーは、管理するホストディレクトリーの絶対パスです。値は、ファイルモジュールへのパラメーターの形式です。値を指定しない場合、ロールは組み込みのデフォルト値を使用します。すべてのホストディレクトリーに使用される 1 つの値を指定する場合は、特殊キー **DEFAULT** を使用します。
podman_firewall	これは dict のリストです。ファイアウォール内でロールが管理するポートを指定します。これは、firewall RHEL システムロールで使用されるものと同じ形式を使用します。

変数	説明
podman_selinux_ports	これは dict のリストです。ロールで使用されるポートの SELinux ポリシーをロールで管理するポートを指定します。これは、selinux RHEL システムロールで使用されるものと同じ形式を使用します。
podman_run_as_user	すべてのルートレスコンテナーに使用するユーザーの名前を指定します。**podman_kube_specs**、**podman_quadlet_specs**、または **podman_secrets** の **run_as_user** で、コンテナー/ユニット/シークレットごとのユーザー名を指定することもできます。  **注記** ユーザーはすでに存在している必要があります。
podman_run_as_group	すべてのルートレスコンテナーに使用するグループの名前を指定します。**podman_kube_specs** または **podman_quadlet_specs** の **run_as_group** を使用して、コンテナーまたはユニットごとのグループ名を指定することもできます。  **注記** グループはすでに存在している必要があります。
podman_systemd_unit_scope	すべての **systemd** ユニットに対してデフォルトで使用する **systemd** スコープを定義します。**podman_kube_specs** および **podman_quadlet_specs** の **systemd_unit_scope** を使用して、コンテナーまたはユニットごとのスコープを指定することもできます。デフォルトでは、ルートレスコンテナーは **user** を使用し、ルートコンテナーは **system** を使用します。
podman_containers_conf	**containers.conf(5)** 設定を dict として定義します。この設定は、**containers.conf.d** ディレクトリー内のドロップインファイルで提供されます。root として実行している場合、**system** 設定が管理されます。**podman_run_as_user** を参照してください。それ以外の場合、**user** 設定が管理されます。ディレクトリーの場所については、**containers.conf** の man ページを参照してください。

変数	説明
podman_registries_conf	**containers-registries.conf(5)** 設定を dict として定義します。この設定は、**registries.conf.d** ディレクトリー内のドロップインファイルで提供されます。root として実行している場合、**system** 設定が管理されます。**podman_run_as_user** を参照してください。それ以外の場合、**user** 設定は管理されます。ディレクトリーの場所については、**registries.conf** の man ページを参照してください。
podman_storage_conf	**containers-storage.conf(5)** 設定を dict として定義します。root として実行している場合、**system** 設定が管理されます。**podman_run_as_user** を参照してください。それ以外の場合、**user** 設定は管理されます。ディレクトリーの場所については、**storage.conf** の man ページを参照してください。
podman_policy_json	**containers-policy.conf(5)** 設定を dict として定義します。root として実行している場合 (**podman_run_as_user** を参照)、**system** 設定が管理されます。それ以外の場合、**user** 設定は管理されます。ディレクトリーの場所については、**policy.json** の man ページを参照してください。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.podman/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/podman/** ディレクトリー

第28章 RHEL システムロールを使用した ANSIBLE による RHEL システムと AD の直接統合

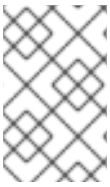
ad_integration システムロールを使用すると、Red Hat Ansible Automation Platform を使用して RHEL システムと Active Directory (AD) の直接統合を自動化できます。

28.1. AD_INTEGRATION システムロール

ad_integration システムロールを使用すると、RHEL システムを Active Directory (AD) に直接接続できます。

ロールは次のコンポーネントを使用します。

- 中央の ID および認証ソースと対話するための SSSD
- 使用可能な AD ドメインを検出し、基盤となる RHEL システムサービス (この場合は SSSD) を設定して、選択した AD ドメインに接続する **realmd**



注記

ad_integration ロールは、Identity Management (IdM) 環境を使用せずに直接 AD 統合を使用するデプロイメント用です。IdM 環境の場合は、**ansible-freeipa** ロールを使用します。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.ad_integration/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/ad_integration/** ディレクトリー
- SSSD を使用して RHEL システムを AD に直接接続

28.2. AD_INTEGRATION RHEL システムロールの変数

ad_integration RHEL システムロールは次のパラメーターを使用します。

ロール変数	説明
ad_integration_realm	参加用の Active Directory レルムまたはドメイン名。
ad_integration_password	マシンをレルムに参加させるときに認証に使用されるユーザーのパスワード。プレーンテキストは使用しないでください。代わりに、Ansible Vault を使用して値を暗号化します。
ad_integration_manage_crypto_policies	**true** の場合、**ad_integration** ロールは、必要に応じて **fedora.linux_system_roles.crypto_policies** を使用します。 デフォルト: **false**

ロール変数	説明
ad_integration_allow_rc4_crypto	**true** の場合、**ad_integration** ロールは、暗号ポリシーを設定して RC4 暗号化を許可します。 この変数を指定すると、**ad_integration_manage_crypto_policies** が自動的に **true** に設定されます。 デフォルト: **false**
ad_integration_timesync_source	システムクロックを同期するタイムソースのホスト名または IP アドレス。この変数を指定すると、**ad_integration_manage_timesync** が **true** に自動的に設定されます。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.ad_integration/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/ad_integration/** ディレクトリー

第29章 POSTGRESQL RHEL システムロールを使用した POSTGRESQL のインストールと設定

システム管理者は、**postgresql** RHEL システムロールを使用して、PostgreSQL サーバーのインストール、設定、管理、起動、パフォーマンスの向上を行うことができます。

29.1. POSTGRESQL ロールの変数

postgresql RHEL システムロールの次の変数を使用して、PostgreSQL サーバーの動作をカスタマイズできます。

postgresql_verison

RHEL 8 および RHEL 9 の管理対象ノードで、PostgreSQL サーバーのバージョンを、リリースおよびサポートされている任意のバージョンの PostgreSQL に設定できます。以下に例を示します。

```
postgresql_version: "13"
```

postgresql_password

必要に応じて、**postgres** データベースのスーパーユーザーのパスワードを設定できます。デフォルトではパスワードは設定されておらず、UNIX ソケットを介して **postgres** システムアカウントからデータベースにアクセスできます。Ansible Vault を使用してパスワードを暗号化することを推奨します。以下に例を示します。

```
postgresql_password: !vault |
      $ANSIBLE_VAULT;1.2;AES256;dev
      ....
```

postgresql_pg_hba_conf

postgresql_pg_hba_conf 変数の内容は、**/var/lib/pgsql/data/pg_hba.conf** ファイル内のデフォルトのアップストリーム設定を置き換えます。以下に例を示します。

```
postgresql_pg_hba_conf:
  - type: local
    database: all
    user: all
    auth_method: peer
  - type: host
    database: all
    user: all
    address: '127.0.0.1/32'
    auth_method: ident
  - type: host
    database: all
    user: all
    address: '::1/128'
    auth_method: ident
```

postgresql_server_conf

postgresql_server_conf 変数の内容は、**/var/lib/pgsql/data/postgresql.conf** ファイルの末尾に追加されます。その結果、デフォルト設定が上書きされます。以下に例を示します。

```
postgresql_server_conf:
  ssl: on
  shared_buffers: 128MB
  huge_pages: try
```

postgresql_ssl_enable

SSL/TLS 接続をセットアップするには、**postgresql_ssl_enable** 変数を **true** に設定します。

```
postgresql_ssl_enable: true
```

次のいずれかの方法を使用して、サーバー証明書と秘密鍵を提供します。

- 既存の証明書と秘密鍵を使用する場合は、**postgresql_cert_name** 変数を使用します。
- 新しい証明書を生成するには、**postgresql_certificates** 変数を使用します。

postgresql_cert_name

独自の証明書と秘密鍵を使用する場合は、**postgresql_cert_name** 変数を使用して証明書名を指定します。証明書ファイルとキーファイルは、同じ名前に **.crt** および **.key** 接尾辞が付いたものとし、同じディレクトリーに保存する必要があります。
たとえば、証明書ファイルが **/etc/certs/server.crt** にあり、秘密鍵が **/etc/certs/server.key** にある場合、**postgresql_cert_name** の値を次のように設定します。

```
postgresql_cert_name: /etc/certs/server
```

postgresql_certificates

postgresql_certificates 変数には、**redhat.rhel_system_roles.certificate** ロールで使用されるのと同じ形式の **dict** の **list** が必要です。PostgreSQL ロールによって設定された PostgreSQL サーバーの証明書を certificate ロールで生成する場合は、**postgresql_certificates** 変数を指定します。次の例では、自己署名証明書 **postgresql_cert.crt** が **/etc/pki/tls/certs/** ディレクトリーに生成されます。デフォルトでは、証明書は自動的に生成されません (**[]**)。

```
postgresql_certificates:
  - name: postgresql_cert
    dns: ['localhost', 'www.example.com']
    ca: self-sign
```

postgresql_input_file

SQL スクリプトを実行するには、**postgresql_input_file** 変数を使用して SQL ファイルへのパスを定義します。

```
postgresql_input_file: "/tmp/mypath/file.sql"
```

postgresql_server_tuning

デフォルトでは、PostgreSQL システムロールにより、システムリソースに基づいたサーバー設定の最適化が有効になります。チューニングを無効にするには、**postgresql_server_tuning** 変数を **false** に設定します。

```
postgresql_server_tuning: false
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.postgresql/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/postgresql/** ディレクトリー

29.2. POSTGRESQL RHEL システムロールの概要

Ansible を使用して PostgreSQL サーバーをインストール、設定、管理、起動するために、**postgresql** RHEL システムロールを使用できます。

postgresql ロールを使用してデータベースサーバー設定を最適化し、パフォーマンスを向上させることもできます。

このロールは、RHEL 8 および RHEL 9 管理対象ノード上で現在リリースされサポートされているバージョンの PostgreSQL をサポートします。

29.3. RHEL システムロールを使用した POSTGRESQL サーバーの設定

postgresql RHEL システムロールを使用して、PostgreSQL サーバーのインストール、設定、管理、起動を行うことができます。

警告

postgresql ロールは、管理対象ホストの **/var/lib/pgsql/data/** ディレクトリー内の PostgreSQL 設定ファイルを置き換えます。以前の設定は、ロール変数で指定された設定に変更され、ロール変数で指定されていない場合は失われます。

前提条件

- コントロールノードと管理対象ノードを準備している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Manage PostgreSQL
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.postgresql
  vars:
    postgresql_version: "13"
```

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.postgresql/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/postgresql/** ディレクトリー
- PostgreSQL の使用