Red Hat Enterprise Linux 7

RHEL 7.9 の 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: 2024-01-18

Red Hat Enterprise Linux 7 RHEL 7.9 の 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、一貫性かつ反復性のある 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 を実行することで、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 システムロールを使用すると、RHEL のメジャーバージョンにまたがる複数の RHEL システムのシステム設定をリモートで管理できます。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。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、RHEL 8 でのコントロールノードの準備 を参照してください。

マネージドノード

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。

インベントリー

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 Linux 8 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

ロール名	ロールの説明	章のタイトル
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
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
microsoft.sql.server	Microsoft SQL Server	microsoft.sql.server Ansible ロールを使用した Microsoft SQL Server の設定
network	ネットワーク	ネットワーク 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
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安全な通信の設定
sshd	SSH サーバー	ssh システムロールを使用した安全な通信の設定
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録画	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	時間同期	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 との VPN 接続の設定

関連情報

- Red Hat Enterprise Linux (RHEL) System Roles
- **rhel-system-roles** パッケージにより提供される **/usr/share/doc/rhel-system-roles/**

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 8 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

- RHEL 7.9 がインストールされている。RHEL のインストールの詳細は、インストールガイドを参照してください。
- 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。
- カスタマーポータルのアカウントで利用可能な場合は、**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]# yum 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): <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
- Updates to using Ansible in RHEL 8.6 and 9.0

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、RHEL 8 でのコントロールノードの準備 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 8 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

3.1. ANSIBLE コレクションの概要

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オプションです。Ansible Collection 形式で RHEL システムロールを使用するのは、従来の RHEL システムロール形式での使用とほぼ同じです。相違点は、Ansible Collections は **完全修飾コレクション名** (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 **コレクション名** で設定されます。使用する **namespace** は **redhat** で、**コレクション名** は **rhel_system_roles** です。したがって、**kernel_settings** ロールの従来の RHEL システムロール形式は (ダッシュを付けて) **rhel-system-roles.kernel_settings** と表示されますが、**kernel_settings** ロールの **fully qualified collection name** コレクションを使用すると、(アンダースコアを付けて) **redhat.rhel_system_roles.kernel_settings** と表示されます。

namespace と **コレクション名** 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。

関連情報

- 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。

3.2. 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

3.3. 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

- 1 つ以上の **管理対象ノード** へのアクセスとパーミッション。
- **コントロールノード** (このシステムから Red Hat Ansible Core は他のシステムを設定) へのアクセスおよびパーミッション。
  コントロールノードでは、
  - **ansible-core** パッケージおよび **rhel-system-roles** パッケージがインストールされている。
  - マネージドノードが記載されているインベントリーファイルがある。

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。

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

インストールが完了すると、ロールは **redhat.rhel_system_roles.<role_name>** として利用できます。また、各ロールのドキュメントは **/usr/share/ansible/collections/ansible_collections/redhat/rhel_system_roles/roles/<role_name>/README.md** で確認できます。

検証手順

インストールを確認するには、localhost で **check** モードで **kernel_settings** ロールを実行します。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、パラメーターはシステムを変更しません。

1. 以下のコマンドを実行します。

```
$ 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。

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

- **ansible-playbook** の man ページ

- **ansible-playbook** コマンドの **-i** オプション

3.4. 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

- 1 つ以上の **管理対象ノード** へのアクセスとパーミッション。
- **コントロールノード** (このシステムから Red Hat Ansible Core は他のシステムを設定) へのアクセスおよびパーミッション。
  コントロールノードでは、
  - **ansible-core** パッケージおよび **rhel-system-roles** パッケージがインストールされている。
  - マネージドノードが記載されているインベントリーファイルがある。

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、ロールは **redhat.rhel_system_roles.<role_name>** として利用できます。また、各ロールのドキュメントは **/usr/share/ansible/collections/ansible_collections/redhat/rhel_system_roles/roles/<role_name>/README.md** で確認できます。

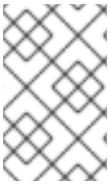
検証手順

インストールを確認するには、localhost で **check** モードで **kernel_settings** ロールを実行します。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、パラメーターはシステムを変更しません。

1. 以下のコマンドを実行します。

```
$ 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。



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

- **ansible-playbook** の man ページ
- **ansible-playbook** コマンドの **-i** オプション

3.5. コレクションを使用したローカルロギングシステムロールの適用

以下の例では、コレクションを使用して Ansible Playbook を準備および適用し、別個のマシンにロギングソリューションを設定します。

前提条件

- rhel-system-roles のコレクション形式は、rpm パッケージまたは Automation Hub のいずれかからインストールされます。

手順

1. 必要なロールを定義する Playbook を作成します。
   a. 新しい YAML ファイルを作成し、これをテキストエディターで開きます。以下に例を示します。

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

   b. 以下の内容を YAML ファイルに挿入します。

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

2. 特定のインベントリーで Playbook を実行します。

   ```
   # ansible-playbook -i inventory-file logging-playbook.yml
   ```

   ここで、

   - **inventory-file** は、インベントリーファイルの名前に置き換えます。
   - **logging-playbook.yml** は、使用する Playbook に置き換えます。

検証手順

1. 設定ファイル **/etc/rsyslog.conf** および **/etc/rsyslog.d** 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. 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です。ログには、logger コマンドを入力したユーザーの名前 (この場合は **root**) が表示されます。

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 7.9 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
User	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. CLI を使用した RHEL MGMT コレクションのインストール

コマンドラインを使用して **rhel_mgmt** コレクションをインストールできます。

前提条件

- **ansible-core** パッケージがインストールされている。

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。

```
# yum install ansible-collection-redhat-rhel_mgmt
```

インストールが完了すると、IPMI モジュールは **redhat.rhel_mgmt** Ansible コレクションで使用できるようになります。

関連情報

- **ansible-playbook** の man ページ

4.3. IPMI_BOOT モジュールの使用例

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

- rhel_mgmt コレクションがインストールされている。

- **python3-pyghmi** パッケージの **pyghmi** ライブラリーが、次のいずれかの場所にインストールされている。
  - Playbook を実行するホスト。
  - マネージドホスト。localhost をマネージドホストとして使用する場合は、代わりに、Playbook を実行するホストに **python3-pyghmi** パッケージをインストールします。
- 制御する IPMI BMC は、Playbook を実行するホスト、またはマネージドホスト (localhost をマネージドホストとして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ます。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モジュールが実行されているホスト (Ansible マネージドホスト) 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. 以下のコンテンツを含む新しい **playbook.yml** ファイルを作成します。

```
---
- name: Sets which boot device will be used on next boot
  hosts: localhost
    tasks:
    - redhat.rhel_mgmt.ipmi_boot:
        name: bmc.host.example.com
          user: admin_user
          password: basics
          bootdev: hd
```

2. localhost に対して Playbook を実行します。

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

その結果、出力は値 success を返します。

4.4. IPMI_POWER モジュールの使用例

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

- rhel_mgmt コレクションがインストールされている。
- **python3-pyghmi** パッケージの **pyghmi** ライブラリーが、次のいずれかの場所にインストールされている。
  - Playbook を実行するホスト。
  - マネージドホスト。localhost をマネージドホストとして使用する場合は、代わりに、Playbook を実行するホストに **python3-pyghmi** パッケージをインストールします。

- 制御する IPMI BMC は、Playbook を実行するホスト、またはマネージドホスト (localhost をマネージドホストとして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ます。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モジュールが実行されているホスト (Ansible マネージドホスト) 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. 以下のコンテンツを含む新しい **playbook.yml** ファイルを作成します。

```
---
- name: Turn the host on
  hosts: localhost
    tasks:
    - redhat.rhel_mgmt.ipmi_power:
        name: bmc.host.example.com
          user: admin_user
          password: basics
          state: on
```

2. Playbook を実行します。

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

出力は値 true を返します。

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

- **redhat.rhel_mgmt** コレクションがインストールされます。
- **python3-pyghmi** パッケージの **pyghmi** ライブラリーが管理対象ホストにインストールされます。管理対象ホストとして localhost を使用する場合は、Playbook を実行するホストに **python3-pyghmi** パッケージをインストールします。
- OOB コントローラーアクセスの詳細。

手順

1. 以下のコンテンツを含む新しい **playbook.yml** ファイルを作成します。

```
---
- name: Get CPU inventory
  hosts: localhost
  tasks:
    - redhat.rhel_mgmt.redfish_info:
        baseuri: "{{ baseuri }}"
        username: "{{ username }}"
        password: "{{ password }}"
        category: Systems
        command: GetCpuInventory
      register: result
```

2. localhost に対して Playbook を実行します。

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

その結果、出力は CPU インベントリーの詳細を返します。

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

- **redhat.rhel_mgmt** コレクションがインストールされます。
- **python3-pyghmi** パッケージの **pyghmi** ライブラリーが管理対象ホストにインストールされます。管理対象ホストとして localhost を使用する場合は、Playbook を実行するホストに **python3-pyghmi** パッケージをインストールします。
- OOB コントローラーアクセスの詳細。

手順

1. 以下のコンテンツを含む新しい **playbook.yml** ファイルを作成します。

```
---
- name: Power on system
  hosts: localhost
  tasks:
    - redhat.rhel_mgmt.redfish_command:
        baseuri: "{{ baseuri }}"
        username: "{{ username }}"
        password: "{{ password }}"
        category: Systems
        command: PowerOn
```

2. localhost に対して Playbook を実行します。

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

その結果、システムの電源が入ります。

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

- **redhat.rhel_mgmt** コレクションがインストールされます。
- **python3-pyghmi** パッケージの **pyghmi** ライブラリーが管理対象ホストにインストールされます。管理対象ホストとして localhost を使用する場合は、Playbook を実行するホストに **python3-pyghmi** パッケージをインストールします。
- OOB コントローラーアクセスの詳細。

手順

1. 以下のコンテンツを含む新しい **playbook.yml** ファイルを作成します。

```
---
- name: "Set BootMode to UEFI"
  hosts: localhost
  tasks:
    - redhat.rhel_mgmt.redfish_config:
        baseuri: "{{ baseuri }}"
        username: "{{ username }}"
        password: "{{ password }}"
        category: Systems
        command: SetBiosAttributes
        bios_attributes:
          BootMode: Uefi
```

2. localhost に対して Playbook を実行します。

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

その結果、システムの起動モードは UEFI に設定されます。

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

Red Hat Ansible の詳しい知識がある経験のあるユーザーは、**kernel_settings** ロールを使用して、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することができます。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 リポジトリーの RPM パッケージとして RHEL のお客様が利用できることに注意してください。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使用しているお客様のコレクションとしても利用できます。

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。

kernel_settings システムロールを使用して、カーネルの自動設定に RHEL システムロールが導入されました。**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/doc/rhel-system-roles/kernel_settings/** ディレクトリーの **README.md** ファイルおよび **README.html** ファイル

- 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

- **root** 権限がある。
- RHEL サブスクリプションの資格を取得して、 **ansible-core** および **rhel-system-roles** パッケージをコントロールマシンにインストールしている。
- マネージドホストのインベントリーが制御マシンに存在し、Ansible から接続できる。

手順

1. 必要に応じて、図の目的で **inventory** ファイルを確認します。

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

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

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

   ファイルは **[testingservers]** グループと他のグループを定義します。これにより、特定のシステムセットに対して Ansible をより効果的に実行できます。

2. 設定ファイルを作成して、Ansible 操作のデフォルトと特権昇格を設定します。

   a. 新しい YAML ファイルを作成し、これをテキストエディターで開きます。以下に例を示します。

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

   b. 以下の内容をファイルに挿入します。

```
[defaults]
inventory = ./inventory

[privilege_escalation]
become = true
```

```
become_method = sudo
become_user = root
become_ask_pass = true
```

[defaults] セクションは、マネージドホストのインベントリーファイルへのパスを指定します。**[privilege_escalation]** セクションでは、指定したマネージドホストのユーザー権限が **root** に移行されることを定義します。これは、カーネルパラメーターを正常に設定するために必要です。Ansible Playbook を実行すると、ユーザーパスワードの入力が求められます。マネージドホストへの接続後に、**sudo** により **root** に自動的に切り替わります。

3. **kernel_settings** ロールを使用する Ansible Playbook を作成します。

   a. 新しい YAML ファイルを作成し、これをテキストエディターで開きます。以下に例を示します。

   ```
   # vi /home/jdoe/<ansible_project_name>/kernel-roles.yml
   ```

   このファイルは Playbook を表し、通常は、**inventory** ファイルから選択した特定のマネージドホストに対して実行される、**プレイ** とも呼ばれるタスクの順序付きリストが含まれます。

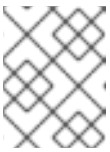
   b. 以下の内容をファイルに挿入します。

```
---
-
  hosts: testingservers
  name: "Configure kernel settings"
  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 キーは任意です。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します。プレイの **hosts** キーは、プレイを実行するホストを指定します。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 セクションは、設定する必要がある、選択したカーネルパラメーター名および値が含まれる変数のリストを表します。

roles キーは、**vars** セクションで説明されているパラメーターおよび値を設定するシステムロールを指定します。



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。

4. 必要に応じて、プレイ内の構文が正しいことを確認します。

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

playbook: kernel-roles.yml
```

以下の例では、Playbook の検証が成功したことを示しています。

5. Playbook を実行します。

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

...

BECOME password:

PLAY [Configure kernel settings]
*********************************************************************************



PLAY RECAP
*********************************************************************************************************
fdoe@192.168.122.226      : ok=10   changed=4    unreachable=0    failed=0    skipped=6
rescued=0    ignored=0
pdoe@192.168.122.98       : ok=10   changed=4    unreachable=0    failed=0    skipped=6
rescued=0    ignored=0
```

Ansible が Playbook を実行する前に、パスワードの入力を求められます。これにより、マネージドホストのユーザーが **root** に切り替わります。これは、カーネルパラメーターの設定に必要です。

recap セクションは、すべてのマネージドホストのプレイが正常に終了したこと (**failed=0**)、および 4 つのカーネルパラメーターが適用されたこと (**changed=4**) を示しています。

6. 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

- RHEL System Roles を使用するための制御ノードと管理対象ノードの準備
- **/usr/share/doc/rhel-system-roles/kernel_settings/** ディレクトリーの **README.html** ファイルおよび **README.md** ファイル
- インベントリーの構築
- Ansible の設定
- 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、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。リモートホスト設定 (**rhc**) 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) サーバーおよび Satellite サーバーに簡単に登録できます。デフォルトでは、**rhc** システムロールを使用してシステムを登録すると、システムは Insights に接続されます。さらに、**rhc** 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 Remediation 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、システムを登録するときに Red Hat Insights に接続します。

前提条件

- 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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。

手順

1. 機密情報を保存する vault を作成します。

```
$ ansible-vault create secrets.yml
New Vault password: password
Confirm New Vault password: password
```

2. **ansible-vault create** コマンドは、暗号化された vault ファイルを作成し、これをエディターで開きます。vault に保存したい機密データを入力します。以下に例を示します。

```
activationKey: activation_key
username: username
password: password
```

3. 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。後で **ansible-vault edit secrets.yml** コマンドを使用して、vault 内のデータを編集できます。

4. オプション: vault の内容を表示します。

```
$ ansible-vault view secrets.yml
```

5. Playbook ファイル (例: ~/**registration.yml)** を作成し、実行するアクションに応じて次のオプションのいずれかを使用します。

   a. 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:
    - secrets.yml
  vars:
    rhc_auth:
      activation_keys:
        keys:
          - "{{ activationKey }}"
     rhc_organization: organizationID
  roles:
    - role: rhel-system-roles.rhc
```

   b. 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:
    - secrets.yml
  vars:
    rhc_auth:
      login:
        username: "{{ username }}"
        password: "{{ password }}"
  roles:
    - role: rhel-system-roles.rhc
```

6. Playbook を実行します。

```
# ansible-playbook ~/registration.yml --ask-vault-pass
```

関連情報

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

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

- 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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。

手順

1. 機密情報を保存する vault を作成します。

```
$ ansible-vault create secrets.yml
New Vault password: password
Confirm New Vault password: password
```

2. **ansible-vault create** コマンドは暗号化されたファイルを作成し、これをエディターで開きます。vault に保存したい機密データを入力します。以下に例を示します。

```
activationKey: activation_key
```

3. 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。
後で **ansible-vault edit secrets.yml** コマンドを使用して、vault 内のデータを編集できます。

4. オプション: vault の内容を表示します。

```
$ ansible-vault view secrets.yml
```

5. Playbook ファイル (例: ~/**registration-sat.yml)** を作成します。

6. アクティベーションキーと組織 ID を使用してシステムを登録するには、~/**registration-sat.yml** で次のテキストを使用します。

```
---
- name: Register to the custom registration server and CDN
  hosts: managed-node-01.example.com
  vars_files:
    - secrets.yml
  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
 roles:
   - role: rhel-system-roles.rhc
```

7. Playbook を実行します。

```
# ansible-playbook ~/registration-sat.yml --ask-vault-pass
```

関連情報

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

7.4. RHC システムロールを使用した登録後の INSIGHTS への接続の無効化

rhc RHEL システムロールを使用してシステムを登録すると、このロールはデフォルトで Red Hat Insights への接続を有効にします。必要ない場合は、**rhc** システムロールを使用して無効にすることができます。

前提条件

- 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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。
- システムはすでに登録されています。

手順

1. Playbook ファイル **(~/dis-insights.yml** など) を作成し、その中に次のコンテンツを追加します。

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

2. Playbook を実行します。

```
# ansible-playbook ~/dis-insights.yml
```

関連情報

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

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

- 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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。
- 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. Playbook ファイルを作成します (例: **~/configure-repos.yml**)。
   a. 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
  vars:
    rhc_repositories:
      - {name: "RepositoryName", state: enabled}
  roles:
    - role: rhel-system-roles.rhc
```

   b. 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 を実行します。

```
# ansible-playbook ~/configure-repos.yml
```

関連情報

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

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

- 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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。
- 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. Playbook ファイル (例: ~/**release.yml**) を作成します。

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

2. Playbook を実行します。

```
# ansible-playbook ~/release.yml
```

関連情報

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

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

- 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は、そのノードに対する **sudo** 権限がある。

- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。

手順

1. 機密情報を保存する vault を作成します。

```
$ ansible-vault create secrets.yml
New Vault password: password
Confirm New Vault password: password
```

2. **ansible-vault create** コマンドは暗号化されたファイルを作成し、これをエディターで開きます。vault に保存したい機密データを入力します。以下に例を示します。

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

3. 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。後で **ansible-vault edit secrets.yml** コマンドを使用して、vault 内のデータを編集できます。

4. オプション: vault の内容を表示します。

```
$ ansible-vault view secrets.yml
```

5. Playbook ファイルを作成します (例: ~/**configure-proxy.yml**)。

   a. 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:
    - secrets.yml
  vars:
    rhc_auth:
      login:
        username: "{{ username }}"
        password: "{{ password }}"
    rhc_proxy:
      hostname: proxy.example.com
      port: 3128
      username: "{{ proxy_username }}"
      password: "{{ proxy_password }}"
  roles:
    - role: rhel-system-roles.rhc
```

   b. 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:
  - secrets.yml
vars:
  rhc_auth:
    login:
      username: "{{ username }}"
      password: "{{ password }}"
   rhc_proxy: {"state":"absent"}
roles:
  - role: rhel-system-roles.rhc
```

6. Playbook を実行します。

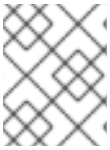
```
# ansible-playbook ~/configure-proxy.yml --ask-vault-pass
```

関連情報

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

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

- 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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。
- システムを登録している。

手順

1. 機密情報を保存する vault を作成します。

```
$ ansible-vault create secrets.yml
New Vault password: password
Confirm New Vault password: password
```

2. **ansible-vault create** コマンドは暗号化されたファイルを作成し、これをエディターで開きます。vault に保存したい機密データを入力します。以下に例を示します。

```
username: username
password: password
```

3. 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。後で **ansible-vault edit secrets.yml** コマンドを使用して、vault 内のデータを編集できます。

4. オプション: vault の内容を表示します。

```
$ ansible-vault view secrets.yml
```

5. Playbook ファイル (例: ~/**auto-update.yml**) を作成し、次のコンテンツを追加します。

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

6. Playbook を実行します。

```
# ansible-playbook ~/auto-update.yml --ask-vault-pass
```

関連情報

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

7.9. RHC RHEL システムロールを使用した INSIGHTS REMEDIATION 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 システムロールで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が確実に修復できるようにします。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 Remediation の詳細は、Red Hat Insights Remediations Guide を参照してください。

前提条件

- 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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。
- Insights 修復が有効になっています。
- システムを登録している。

手順

1. 修復を有効にするには、Playbook ファイル (例: ~/**remediation.yml**) を作成します。

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

2. Playbook を実行します。

```
# ansible-playbook ~/remediation.yml
```

関連情報

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

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

- 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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。

手順

1. 機密情報を保存する vault を作成します。

```
$ ansible-vault create secrets.yml
New Vault password: password
Confirm New Vault password: password
```

2. **ansible-vault create** コマンドは暗号化されたファイルを作成し、これをエディターで開きます。vault に保存したい機密データを入力します。以下に例を示します。

```
username: username
password: password
```

3. 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。後で **ansible-vault edit secrets.yml** コマンドを使用して、vault 内のデータを編集できます。

4. オプション: vault の内容を表示します。

```
$ ansible-vault view secrets.yml
```

5. Playbook ファイル (例: ~/**tags.yml**) を作成し、次のコンテンツを追加します。

```
---
- name: Creating tags
  hosts: managed-node-01.example.com
  vars_files:
    - secrets.yml
  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
  roles:
    - role: rhel-system-roles.rhc
```

6. Playbook を実行します。

```
# ansible-playbook ~/remediation.yml --ask-vault-pass
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.rhc/README.md** ファイル
- 詳細は、System Filtering and groups Red Hat Insights を参照してください。

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

- 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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。
- システムはすでに登録されています。

手順

1. 登録を解除するには、Playbook ファイル (例: **~/unregister.yml**) を作成し、次のコンテンツを追加します。

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

2. Playbook を実行します。

```
# ansible-playbook ~/unregister.yml
```

関連情報

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

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. ネットワーク 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。

たとえば、以下の手順では、以下の設定で **enp7s0** デバイスの NetworkManager 接続プロファイルを作成します。

- 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**

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

- 制御ノードと管理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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。
- 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. **~/ethernet-static-IP.yml** などの Playbook ファイルを次の内容で作成します。

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
  - name: Configure an Ethernet connection with static IP
    include_role:
```

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

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

2. Playbook を実行します。

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

関連情報

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

8.2. ネットワーク 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=
```

たとえば、以下の手順では、PCI ID **0000:00:0[1-3].0** 式に一致するが、**0000:00:02.0** は一致しないデバイスに対して、以下の設定で NetworkManager 接続プロファイルを作成します。

- 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**

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

- 制御ノードと管理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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。
- 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. **~/ethernet-static-IP.yml** などの Playbook ファイルを次の内容で作成します。

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

この例の **match** パラメーターは、Ansible が PCI ID **0000:00:0[1-3].0** に一致するデバイスに再生を適用するが、**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 を実行します。

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

関連情報

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

8.3. ネットワーク 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。

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

- 制御ノードと管理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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。
- 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. **~/ethernet-dynamic-IP.yml** などの Playbook ファイルを次の内容で作成します。

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
  - name: Configure an Ethernet connection with dynamic IP
    include_role:
      name: rhel-system-roles.network

    vars:
      network_connections:
        - name: enp7s0
          interface_name: enp7s0
          type: ethernet
          autoconnect: yes
          ip:
```

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

2. Playbook を実行します。

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

関連情報

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

8.4. ネットワーク 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=
```

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

- 制御ノードと管理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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。
- 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. **~/ethernet-dynamic-IP.yml** などの Playbook ファイルを次の内容で作成します。

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
  - name: Configure an Ethernet connection with dynamic IP
    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
```

この例の **match** パラメーターは、Ansible が PCI ID **0000:00:0[1-3].0** に一致するデバイスに再生を適用するが、**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 を実行します。

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

関連情報

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

8.5. ネットワーク RHEL システムロールを使用した VLAN タギングの設定

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。

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

- 制御ノードと管理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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。

手順

1. ~/**vlan-ethernet.yml** などの Playbook ファイルを次の内容で作成します。

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
  - name: Configure a VLAN that uses an Ethernet connection
    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
```

   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。

2. Playbook を実行します。

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

関連情報

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

8.6. ネットワーク 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、Linux ブリッジを設定できます。たとえば、2 つのイーサネットデバイスを使用するネットワークブリッジを設定し、IPv4 アドレスおよび IPv6 アドレス、デフォルトゲートウェイ、および DNS 設定を設定します。

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。

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

- 制御ノードと管理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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。
- 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. **~/bridge-ethernet.yml** などの Playbook ファイルを次の内容で作成します。

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

2. Playbook を実行します。

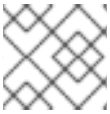
```
# ansible-playbook ~/bridge-ethernet.yml
```

関連情報

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

8.7. ネットワーク 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 の RHEL システムロールを使用して、Linux ボンディングを設定できます。たとえば、2 つのイーサネットデバイスを使用する active-backup モードでネットワークボンディングを設定し、IPv4 アドレスおよび IPv6 アドレス、デフォルトゲートウェイ、および DNS 設定を設定します。



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。

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

- 制御ノードと管理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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。
- 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. ~/**bond-ethernet.yml** などの Playbook ファイルを次の内容で作成します。

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

2. Playbook を実行します。

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

関連情報

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

8.8. ネットワーク 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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。
- **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. **~/IPoIB.yml** などの Playbook ファイルを次の内容で作成します。

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
  - name: Configure IPoIB
    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 を実行します。

```
# ansible-playbook ~/IPoIB.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

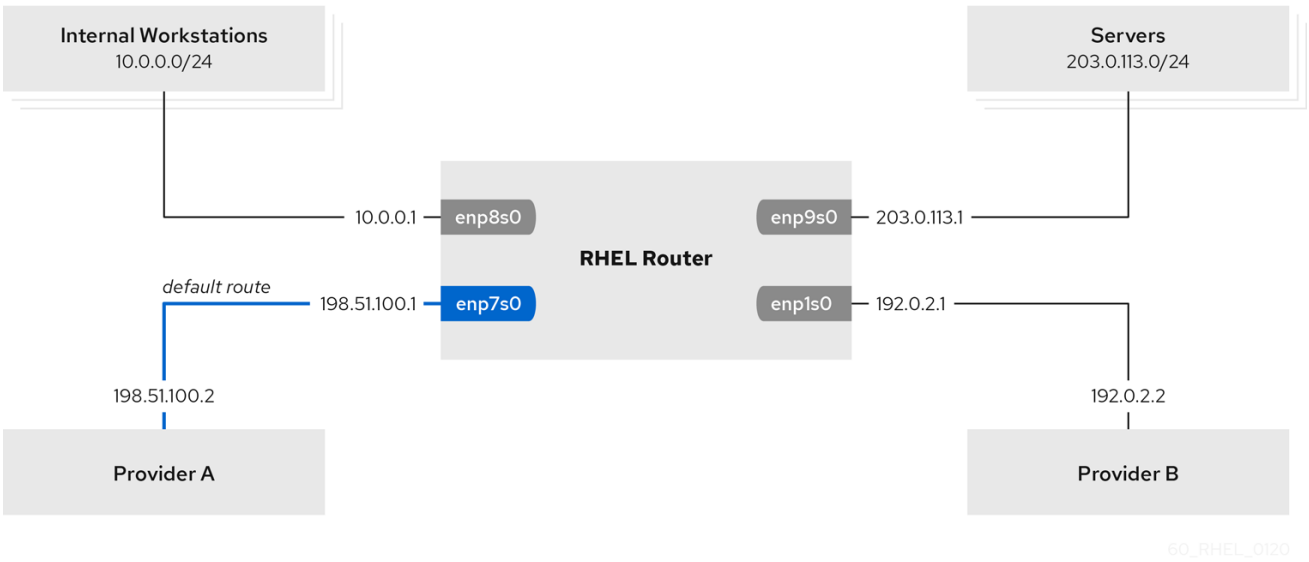
- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル

8.9. ネットワーク 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、RHEL **network** システムロールを使用できます。Ansible コントロールノードで以下の手順を実行します。

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

- 制御ノードと管理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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。
- 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. ~/**pbr.yml** などの Playbook ファイルを次の内容で作成します。

```
---
- name: Configuring policy-based routing
  hosts: managed-node-01.example.com
  tasks:
  - name: Routing traffic from a specific subnet to a different default gateway
    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 を実行します。

```
# ansible-playbook ~/pbr.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。

   ```
   # yum 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。

   ```
   # yum 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

8.10. ネットワーク 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 規格を使用してクライアントを認証するイーサネット接続の作成を自動化できます。たとえば、Ansible Playbook を実行して、以下の設定で **enp1s0** インターフェイスのイーサネット接続をリモートで追加します。

- 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

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

- 制御ノードと管理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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象のノードまたは管理対象のノードのグループは、Ansible インベントリーファイルにリストされています。
- 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. **~/enable-802.1x.yml** などの Playbook ファイルを次の内容で作成します。

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

    - name: Copy client certificate for 802.1X authentication
```

```
    copy:
      src: "/srv/data/client.crt"
      dest: "/etc/pki/tls/certs/client.crt"

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

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

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

2. Playbook を実行します。

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

関連情報

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

8.11. ネットワーク RHEL システムロールを使用した既存の接続でのデフォルトゲートウェイの設定

network の 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 の RHEL システムロールを使用する再生を実行すると、設定値が再生で指定されたものと一致しない場合に、システムロールが同じ名前の既存の接続プロファイルを上書きします。したがって、IP 設定がすでに存在している場合でも、プレイでネットワーク接続プロファイルの設定全体を指定する必要があります。それ以外の場合は、ロールはこれらの値をデフォルト値にリセットします。

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**

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

- 制御ノードと管理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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。

手順

1. **~/ethernet-connection.yml** などの Playbook ファイルを次の内容で作成します。

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
  - name: Configure an Ethernet connection with static IP and default gateway
    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 を実行します。

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

関連情報

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

8.12. ネットワーク 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 の RHEL システムロールを使用する再生を実行すると、設定値が再生で指定されたものと一致しない場合に、システムロールが同じ名前の既存の接続プロファイルを上書きします。したがって、IP 設定がすでに存在している場合でも、プレイでネットワーク接続プロファイルの設定全体を指定する必要があります。それ以外の場合は、ロールはこれらの値をデフォルト値にリセットします。

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**

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

- 制御ノードと管理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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。

手順

1. **~/add-static-routes.yml** などの Playbook ファイルを次の内容で作成します。

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
  - name: Configure an Ethernet connection with static IP and additional routes
    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
```

2. Playbook を実行します。

```
# ansible-playbook ~/add-static-routes.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

8.13. ネットワーク 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 の 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 の RHEL システムロールを使用する再生を実行すると、設定値が再生で指定されたものと一致しない場合に、システムロールが同じ名前の既存の接続プロファイルを上書きします。したがって、IP 設定などがすでに存在している場合でも、常にネットワーク接続プロファイルの設定全体をプレイで指定します。それ以外の場合は、ロールはこれらの値をデフォルト値にリセットします。

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** 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

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

- 制御ノードと管理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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。

手順

1. Playbook ファイル (例: ~/**configure-ethernet-device-with-ethtool-features.yml**) を次の内容で作成します。

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
  - name: Configure an Ethernet connection with ethtool features
    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
```

2. Playbook を実行します。

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

関連情報

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

8.14. ネットワーク 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

<pre>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</pre>
	<pre>vars:
	  network_connections:
	    - name: enp7s0
	      interface_name: enp7s0
	      type: ethernet
	      autoconnect: yes
	      ip:
	        dhcp4: yes
	        auto6: yes
	      state: up</pre>

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	通常のPlaybook

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

```
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

第9章 システムロールを使用した FIREWALLD の設定

firewall 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、システムロールは **firewalld** パラメーターをマネージドノードに即座に適用し、再起動後も維持されます。

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。

firewalld サービスの自動設定に、RHEL システムロールからの **rhel-system-roles.firewall** ロールが導入されました。**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。

firewalld パラメーターを自動化された方法で 1 つ以上のシステムに適用するには、Playbook で **firewall** 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/doc/rhel-system-roles/firewall/** ディレクトリーの **README.md** ファイルおよび **README.html** ファイル
- Playbook の使用
- インベントリーの構築方法

9.2. ファイアウォール 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。**previous:replaced** パラメーターを変数リストに追加すると、システムロールは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

- 制御ノードと管理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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。

手順

1. **~/reset-firewalld.yml** などの Playbook ファイルを次の内容で作成します。

```
---
- name: Reset firewalld example
  hosts: managed-node-01.example.com
  tasks:
  - name: Reset firewalld
    include_role:
      name: rhel-system-roles.firewall

    vars:
      firewall:
        - previous: replaced
```

2. Playbook を実行します。

```
# ansible-playbook ~/configuring-a-dmz.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**
- **ansible-playbook(1)**
- **firewalld(1)**

9.3. 別のローカルポートへの着信トラフィックの転送

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

- 制御ノードと管理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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。

手順

1. **~/port_forwarding.yml** などの Playbook ファイルを次の内容で作成します。

```
---
- name: Configure firewalld
  hosts: managed-node-01.example.com
  tasks:
  - name: Forward incoming traffic on port 8080 to 443
    include_role:
      name: rhel-system-roles.firewall

    vars:
      firewall:
        - { forward_port: 8080/tcp;443;, state: enabled, runtime: true, permanent: true }
```

2. Playbook を実行します。

```
# ansible-playbook ~/port_forwarding.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**

9.4. システムロールを使用したポートの設定

RHEL **firewall** システムロールを使用すると、着信トラフィックに対してローカルファイアウォールでポートを開くか閉じて、再起動後に新しい設定を永続化できます。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

- 制御ノードと管理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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。

手順

1. **~/opening-a-port.yml** などの Playbook ファイルを次の内容で作成します。

```
---
- name: Configure firewalld
  hosts: managed-node-01.example.com
  tasks:
  - name: Allow incoming HTTPS traffic to the local host
    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 を実行します。

```
# ansible-playbook ~/opening-a-port.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**

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

システム管理者は、**firewall** システムロールを使用して、**enp1s0** インターフェイスで **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

- 制御ノードと管理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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。

手順

1. **~/configuring-a-dmz.yml** などの Playbook ファイルを次の内容で作成します。

```
---
- 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
    include_role:
      name: rhel-system-roles.firewall

    vars:
      firewall:
        - zone: dmz
          interface: enp1s0
          service: https
          state: enabled
          runtime: true
          permanent: true
```

2. Playbook を実行します。

```
# ansible-playbook ~/configuring-a-dmz.yml
```

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**

第10章 システムロールの POSTFIX ロールの変数

postfix ロール変数により、ユーザーは **postfix** Mail Transfer Agent (MTA) をインストール、設定、および起動できます。

以下のロール変数がこのセクションで定義されています。

- **postfix_conf**: 対応している **postfix** 設定パラメーターすべてのキー/値のペアが含まれます。初期設定では、この **postfix_conf** には値が設定されていません。

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

シナリオで既存の設定を削除し、必要な設定を **Postfix** のクリーンインストールの上に適用する必要がある場合は、**postfix_conf** ディクショナリー内で **previous: replaced** オプションを指定します。

previous: replaced オプション:

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

- **postfix_check**: 設定変更を検証するために、**postfix** の起動前に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**: 設定のバックアップコピーが 1 つ作成されているかどうかを決定します。デフォルトでは、**postfix_backup** の値は false です。

以前のバックアップを上書きする場合は、以下のコマンドを実行します。

```
# *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。

以下に例を示します。

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

- **postfix_backup_multiple**: ロールが設定のタイムスタンプ付きバックアップコピーを作成するかどうかを決定します。

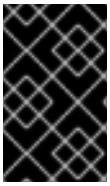
複数のバックアップコピーを保持するには、次のコマンドを実行します。

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

デフォルトでは、**postfix_backup_multiple** の値は true です。**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**: **postfix** ロールを **firewall** ロールと統合して、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**: **postfix** ロールを **selinux** ロールと統合して、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。



重要

設定パラメーターは削除できません。**postfix** ロールを実行する前に、**postfix_conf** を必要なすべての設定パラメーター設定し、ファイルモジュールを使用して **/etc/postfix/main.cf** を削除します。

10.1. 関連情報

- **/usr/share/doc/rhel-system-roles/postfix/README.md**

第11章 システムロールを使用した SELINUX の設定

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集合です。**selinux** システムロールでは、以下の操作が可能になります。

- 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

以下の表は、**selinux** システムロールで利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。

selinux ロール変数の詳細は、**rhel-system-roles** パッケージをインストールし、**/usr/share/doc/rhel-system-roles/selinux/** ディレクトリーの **README.md** または **README.html** ファイルを参照してください。

関連情報

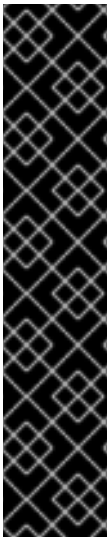
- RHEL システムロールの概要

11.2. SELINUX システムロールを使用した、複数のシステムでの SELINUX 設定の適用

以下の手順に従って、検証した SELinux 設定を使用して Ansible Playbook を準備し、適用します。

前提条件

- 1 つまたは複数の **管理対象ノード** へのアクセスとパーミッション (**selinux** システムロールで設定するシステム)。
- **コントロールノード** (このシステムから Red Hat Ansible Core は他のシステムを設定) へのアクセスおよびパーミッション。
  コントロールノードでは、
  - **ansible-core** パッケージおよび **rhel-system-roles** パッケージがインストールされている。
  - マネージドノードが記載されているインベントリーファイルがある。



重要

RHEL 8.0-8.5 では、別の Ansible リポジトリーへのアクセス権を指定されており、Ansible をベースにする自動化用の Ansible Engine 2.9 が含まれています。Ansible Engine には、**ansible**、**ansible-playbook** などのコマンドラインユーティリティー、**docker** や **podman** などのコネクター、プラグインとモジュールが多く含まれています。Ansible Engine を入手してインストールする方法の詳細は、ナレッジベースのアーティクル記事 How to download and install Red Hat Ansible Engine を参照してください。

RHEL 8.6 および 9.0 では、Ansible Core (**ansible-core** パッケージとして提供) が導入されました。これには、Ansible コマンドラインユーティリティー、コマンド、およびビルトイン Ansible プラグインのセットが含まれています。RHEL は、AppStream リポジトリーを介してこのパッケージを提供し、サポート範囲は限定的です。詳細については、ナレッジベースの Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories を参照してください。

- マネージドノードが記載されているインベントリーファイルがある。

手順

1. Playbook を準備します。ゼロから開始するか、**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. **host1**、**host2** および **host3** システムで Playbook を実行します。

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

関連情報

- 詳細は、**rhel-system-roles** パッケージをインストールして、**/usr/share/doc/rhel-system-roles/selinux/** ディレクトリーおよび **/usr/share/ansible/roles/rhel-system-roles.selinux/** ディレクトリーを参照してください。

第12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** 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。

12.1. LOGGING システムロール

Logging システムロールを使用すると、ローカルおよびリモートホストにロギング設定をデプロイできます。

Logging システムロールを 1 つ以上のシステムに適用するには、**Playbook** でロギング設定を定義します。Playbook は、1 つ以上の play のリストです。Playbook は YAML 形式で表現され、人が判読できるようになっています。Playbook の詳細は、Ansible ドキュメントの Working with playbooks を参照してください。

Playbook に従って設定するシステムのセットは、**インベントリーファイル** で定義されます。インベントリーの作成および使用に関する詳細は、Ansible ドキュメントの How to build your inventory を参照してください。

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**
- ネットワーク上で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 システムロールでは、シナリオに合わせて入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。

12.2. LOGGING システムロールのパラメーター

logging システムロール Playbook では、**logging_inputs** パラメーターで入力を、**logging_outputs** パラメーターで出力を、そして **logging_flows** パラメーターで入力と出力の関係を定義します。**Logging** システムロールは、ロギングシステムの追加設定オプションで、上記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** システムロールで利用可能な唯一のロギングシステムは **Rsyslog** 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、変数は **firewall** ロールを使用して、**logging** ロール内からの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、変数は **selinux** ロールを使用して、**logging** ロール内からポートアクセスを自動的に管理します。

関連情報

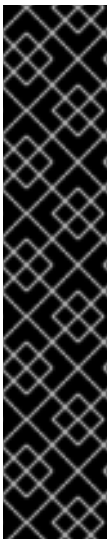
- **rhel-system-roles** パッケージでインストールされたドキュメント (**/usr/share/ansible/roles/rhel-system-roles.logging/README.html**)

12.3. 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

- Logging システムロールを設定する **管理対象ノード** 1 つ以上へのアクセスおよびパーミッション。
- **コントロールノード** (このシステムから Red Hat Ansible Core は他のシステムを設定) へのアクセスおよびパーミッション。
  コントロールノードでは、
  - **ansible-core** パッケージおよび **rhel-system-roles** パッケージがインストールされている。



重要

RHEL 8.0-8.5 では、別の Ansible リポジトリーへのアクセス権を指定されており、Ansible をベースにする自動化用の Ansible Engine 2.9 が含まれています。Ansible Engine には、**ansible**、**ansible-playbook** などのコマンドラインユーティリティー、**docker** や **podman** などのコネクター、プラグインとモジュールが多く含まれています。Ansible Engine を入手してインストールする方法の詳細は、ナレッジベースのアーティクル記事 How to download and install Red Hat Ansible Engine を参照してください。

RHEL 8.6 および 9.0 では、Ansible Core (**ansible-core** パッケージとして提供) が導入されました。これには、Ansible コマンドラインユーティリティー、コマンド、およびビルトイン Ansible プラグインのセットが含まれています。RHEL は、AppStream リポジトリーを介してこのパッケージを提供し、サポート範囲は限定的です。詳細については、ナレッジベースの Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories を参照してください。

- マネージドノードが記載されているインベントリーファイルがある。

注記

デプロイメント時にシステムロールが **rsyslog** をインストールするため、**rsyslog** パッケージをインストールする必要はありません。

手順

1. 必要なロールを定義する Playbook を作成します。
   a. 新しい YAML ファイルを作成し、これをテキストエディターで開きます。以下に例を示します。

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

   b. 以下の内容を挿入します。

   ```
   ---
   - name: Deploying basics input and implicit files output
     hosts: all
   ```

```
roles:
  - rhel-system-roles.logging
vars:
  logging_inputs:
    - name: system_input
      type: basics
  logging_outputs:
    - name: files_output
      type: files
  logging_flows:
    - name: flow1
      inputs: [system_input]
      outputs: [files_output]
```

2. 特定のインベントリーで Playbook を実行します。

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

詳細は以下のようになります。

- **inventory-file** はインベントリーファイルに置き換えます。
- **logging-playbook.yml** は、使用する Playbook に置き換えます。

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
```

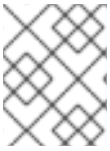
`hostname` はクライアントシステムのホスト名です。ログには、logger コマンドを入力したユーザーのユーザー名 (この場合は **root**) が含まれていることに注意してください。

12.4. ローカルの LOGGING システムロールでのログのフィルタリング

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

- Logging システムロールを設定する **管理対象ノード** 1 つ以上へのアクセスおよびパーミッション。
- **コントロールノード** (このシステムから Red Hat Ansible Core は他のシステムを設定) へのアクセスおよびパーミッション。
  コントロールノードでは、
  - Red Hat Ansible Core がインストールされている。
  - **rhel-system-roles** パッケージがインストールされている。
  - マネージドノードが記載されているインベントリーファイルがある。



注記

デプロイメント時にシステムロールが **rsyslog** をインストールするため、**rsyslog** パッケージをインストールする必要はありません。

手順

1. 以下の内容を含む新しい **playbook.yml** ファイルを作成します。

```
---
- name: Deploying files input and configured files output
  hosts: all
  roles:
    - linux-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. オプション: Playbook の構文を確認します。

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

3. インベントリーファイルで Playbook を実行します。

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

- **rhel-system-roles** パッケージでインストールされたドキュメント (**/usr/share/ansible/roles/rhel-system-roles.logging/README.html**)

12.5. 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

- Logging システムロールを設定する **管理対象ノード** 1 つ以上へのアクセスおよびパーミッション。
- **コントロールノード** (このシステムから Red Hat Ansible Core は他のシステムを設定) へのアクセスおよびパーミッション。
  コントロールノードでは、

- **ansible-core** パッケージおよび **rhel-system-roles** パッケージがインストールされている。
- マネージドノードが記載されているインベントリーファイルがある。

注記

デプロイメント時にシステムロールが **rsyslog** をインストールするため、**rsyslog** パッケージをインストールする必要はありません。

手順

1. 必要なロールを定義する Playbook を作成します。
   a. 新しい YAML ファイルを作成し、これをテキストエディターで開きます。以下に例を示します。

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

   b. 以下の内容をファイルに挿入します。

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

- name: Deploying basics input and forwards output
  hosts: clients
  roles:
    - rhel-system-roles.logging
  vars:
    logging_inputs:
      - name: basic_input
        type: basics
    logging_outputs:
      - name: forward_output0
        type: forwards
        severity: info
        target: _host1.example.com_
```

```
      udp_port: 601
    - name: forward_output1
      type: forwards
      facility: mail
      target: _host1.example.com_
      tcp_port: 601
  logging_flows:
    - name: flows0
      inputs: [basic_input]
      outputs: [forward_output0, forward_output1]

[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. サーバーおよびクライアントをリスト表示するインベントリーファイルを作成します。
   a. 新しいファイルを作成してテキストエディターで開きます。以下に例を示します。

   ```
   # vi inventory.ini
   ```

   b. 以下のコンテンツをインベントリーファイルに挿入します。

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

   詳細は以下のようになります。

   - **host1.example.com** はロギングサーバーです。
   - **host2.example.com** はロギングクライアントです。

3. インベントリーで Playbook を実行します。

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

詳細は以下のようになります。

- **inventory.ini** はインベントリーファイルに置き換えます。
- **logging-playbook.yml** は作成した Playbook に置き換えます。

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/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 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です。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

- RHEL System Roles を使用するための制御ノードと管理対象ノードの準備
- **rhel-system-roles** パッケージでインストールされたドキュメント (**/usr/share/ansible/roles/rhel-system-roles.logging/README.html**)
- RHEL システムロール のナレッジベース記事

12.6. 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安全に通信するために設計された暗号プロトコルです。

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。

12.6.1. TLS を使用したクライアントロギングの設定

logging システムロールを使用して、ローカルマシンにログインしている RHEL システムでロギングを設定し、Ansible Playbook を実行して、ログを TLS で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、Ansible インベントリーの client グループ内の全ホストに 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。

前提条件

- TLS を設定する管理対象ノードで Playbook の実行権限がある。
- マネージドノードがコントロールノードのインベントリーファイルに記載されている。
- **ansible** パッケージおよび **rhel-system-roles** パッケージがコントロールノードにインストールされている。

手順

1. 以下の内容を含む **playbook.yml** ファイルを作成します。

```
---
- name: Deploying files input and forwards output with certs
  hosts: clients
  roles:
    - rhel-system-roles.logging
  vars:
    logging_pki_files:
      - ca_cert_src: /local/path/to/ca_cert.pem
        cert_src: /local/path/to/cert.pem
        private_key_src: /local/path/to/key.pem
    logging_inputs:
      - name: input_name
        type: 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_pki_files

このパラメーターを使用して、TLS を設定し、**ca_cert_src**、**cert_src** および **private_key_src** パラメーターを指定する必要があります。

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。

tls

このパラメーターを使用すると、ネットワーク経由でログを安全に転送できるようになります。セキュアなラッパーが必要ない場合は、**tls: true** と設定できます。

2. Playbook の構文を確認します。

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

3. インベントリーファイルで Playbook を実行します。

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

12.6.2. TLS を使用したサーバーロギングの設定

logging システムロールを使用して、RHEL システムのログインをサーバーとして設定し、Ansible Playbook を実行して TLS で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 TLS を設定します。

前提条件

- TLS を設定する管理対象ノードで Playbook の実行権限がある。
- マネージドノードがコントロールノードのインベントリーファイルに記載されている。
- **ansible** パッケージおよび **rhel-system-roles** パッケージがコントロールノードにインストールされている。

手順

1. 以下の内容を含む **playbook.yml** ファイルを作成します。

```
---
- name: Deploying remote input and remote_files output with certs
  hosts: server
  roles:
    - rhel-system-roles.logging
  vars:
    logging_pki_files:
```

```
      - ca_cert_src: /local/path/to/ca_cert.pem
        cert_src: /local/path/to/cert.pem
        private_key_src: /local/path/to/key.pem
    logging_inputs:
      - name: input_name
        type: remote
        tcp_ports: 514
        tls: true
        permitted_clients: ['clients.example.com']
    logging_outputs:
      - name: output_name
        type: remote_files
        remote_log_path: /var/log/remote/%FROMHOST%/%PROGRAMNAME:::secpath-
replace%.log
        async_writing: true
        client_count: 20
        io_buffer_size: 8192
    logging_flows:
      - name: flow_name
        inputs: [input_name]
        outputs: [output_name]
```

Playbook は以下のパラメーターを使用します。

logging_pki_files

このパラメーターを使用して、TLS を設定し、**ca_cert_src**、**cert_src** および **private_key_src** パラメーターを指定する必要があります。

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。

tls

このパラメーターを使用すると、ネットワーク経由でログを安全に転送できるようになります。セキュアなラッパーが必要ない場合は、**tls: true** と設定できます。

2. Playbook の構文を確認します。

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

3. インベントリーファイルで Playbook を実行します。

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

12.7. 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** 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することができます。

12.7.1. RELP を使用したクライアントロギングの設定

logging システムロールを使用して、ローカルマシンにログインしている RHEL システムでロギングを設定し、Ansible Playbook を実行して、ログを RELP で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、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

- RELP を設定する管理対象ノードで Playbook の実行権限がある。
- マネージドノードがコントロールノードのインベントリーファイルに記載されている。
- **ansible** パッケージおよび **rhel-system-roles** パッケージがコントロールノードにインストールされている。

手順

1. 以下の内容を含む **playbook.yml** ファイルを作成します。

```
---
- name: Deploying basic input and relp output
  hosts: clients
  roles:
    - rhel-system-roles.logging
  vars:
    logging_inputs:
      - name: basic_input
        type: basics
```

```
    logging_outputs:
      - name: relp_client
        type: relp
        target: _logging.server.com_
        port: 20514
        tls: true
        ca_cert: _/etc/pki/tls/certs/ca.pem_
        cert: _/etc/pki/tls/certs/client-cert.pem_
        private_key: _/etc/pki/tls/private/client-key.pem_
        pki_authmode: name
        permitted_servers:
          - '*.server.example.com'
    logging_flows:
      - name: _example_flow_
        inputs: [basic_input]
        outputs: [relp_client]
```

Playbook は、以下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**) を、コントロールノードから転送するマネージドノードの宛先として使用します。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には、ファイルはロギング設定の前にデフォルトのパスを配置する必要があります。
  - 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マネージド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 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 が指定している場合は、その場所にコピーされます。
- **cert_src**: ローカル証明書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。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には、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. オプション: Playbook の構文を確認します。

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

3. Playbook を実行します。

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

12.7.2. RELP を使用したサーバーログの設定

logging システムロールを使用して、RHEL システムのログインをサーバーとして設定し、Ansible Playbook を実行して RELP で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

- RELP を設定する管理対象ノードで Playbook の実行権限がある。
- マネージドノードがコントロールノードのインベントリーファイルに記載されている。
- **ansible** パッケージおよび **rhel-system-roles** パッケージがコントロールノードにインストールされている。

手順

1. 以下の内容を含む **playbook.yml** ファイルを作成します。

```
---
- name: Deploying remote input and remote_files output
  hosts: server
  roles:
    - rhel-system-roles.logging
  vars:
    logging_inputs:
      - name: relp_server
        type: relp
        port: 20514
        tls: true
        ca_cert: _/etc/pki/tls/certs/ca.pem_
        cert: _/etc/pki/tls/certs/server-cert.pem_
```

```
      private_key: _/etc/pki/tls/private/server-key.pem_
      pki_authmode: name
      permitted_clients:
        - '_*example.client.com_'
  logging_outputs:
    - name: _remote_files_output_
      type: _remote_files_
  logging_flows:
    - name: _example_flow_
      inputs: _relp_server_
      outputs: _remote_files_output_
```

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**) を、コントロールノードから転送するマネージドノードの宛先として使用します。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には、ファイルはロギング設定の前にデフォルトのパスを配置する必要があります。
  - 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マネージド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 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 が指定している場合は、その場所にコピーされます。
- **cert_src**: ローカル証明書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。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には、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. オプション: Playbook の構文を確認します。

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

3. Playbook を実行します。

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

12.8. 関連情報

- RHEL System Roles を使用するための制御ノードと管理対象ノードの準備
- **rhel-system-roles** パッケージでインストールされたドキュメントは、**/usr/share/ansible/roles/rhel-system-roles.logging/README.html** にあります。
- RHEL システムロール
- **ansible-playbook(1)** の man ページ。

第13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 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グを設定できます。

13.1. JOURNALD RHEL システムロールの変数

journald 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。

ロール変数	説明
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)** の man ページで説明されているデフォルトのサイジング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

- **journald.conf(5)** の man ページ

13.2. JOURNALD システムロールを使用した永続ログの設定

システム管理者は、**journald** 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** 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

- 制御ノードと管理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は、そのノードに対する **sudo** 権限がある。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリーファイルにリストされている。

手順

1. 以下のコンテンツを含む新しい **playbook.yml** ファイルを作成します。

```
---
- hosts: all
  vars:
    journald_persistent: true
    journald_max_disk_size: 2048
    journald_per_user: true
    journald_sync_interval: 1
  roles:
    - linux-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. オプション: Playbook の構文を確認します。

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

3. インベントリーファイルで Playbook を実行します。

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

13.3. 関連情報

- **journald.conf(5)** の man ページ
- **ansible-playbook(1)** の man ページ

第14章 SSH および SSHD RHEL システムロールを使用した安全な通信の設定

管理者は、**sshd** システムロールを使用して SSH サーバーを設定し、 **ssh** システムロールを使用して任意数の RHEL システムに同時に同じ設定の SSH クライアントを Red Hat Ansible Automation Platform で設定できます。

14.1. SSH SERVER のシステムロール変数

sshd システムロール Playbook では、設定と制限に応じて、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、システムロールではデフォルト値が適用されません。代わりに、**sshd** dict または **sshd_Key** 変数のいずれかを使用して、設定のデフォルト値をすべて指定します。デフォルトは **False** です。

sshd_manage_service

False に設定すると、サービスはマネージドではなくなるので、起動時に有効化されず、起動または再読み込みされません。Ansible サービスモジュールが現在 AIX で **enabled** になっていないため、コンテナーまたは AIX 内で実行する時以外はデフォルトで **True** に設定されます。

sshd_allow_reload

False に設定すると、設定の変更後に **sshd** は再読み込みされません。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、**sshd_manage_service** と同じ値にデフォルト設定されます。ここで、**sshd_manage_service** はデフォルトで **False** に設定されますが、**sshd_allow_reload** はデフォルトで **True** に設定されます。

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 つ目のインスタンスを設定し、**sshd_service** 変数も変更しない限り、**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 dict。以下に例を示します。

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

sshd_OptionName

dict の代わりに、**sshd_** 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、**sshd** dict の値を上書きします。以下に例を示します。

```
sshd_Compression: no
```

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

dict のリスト、または Match セクションの dict のみ。これらの変数は、**sshd** dict で定義されている一致するブロックを上書き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 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

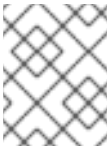
このロールは、これらの変数を使用して生成する **openssh** 設定ファイルの所有権およびパーミッションを設定できます。

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、ロールがデプロイメント段階で使用され、サービスが最初の試行で起動できるようにする場合に便利です。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 ベースのシステムでは、この変数は **sshd** サービスに関する追加情報を設定します。**true** に設定すると、このロールは以下の設定に基づいて **/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

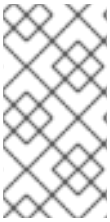
RHEL では、**true** に設定すると、この変数はシステム全体の暗号化ポリシーを上書きします。デフォルトは **false** です。

sshd_sysconfig_use_strong_rng

RHEL ベースのシステムでは、この変数により、**sshd** は、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することができます。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。

14.2. 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 を実行することで複数の 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

- 1 つ以上の **管理対象ノード** (**sshd** システムロールで設定するシステム) へのアクセスおよびパーミッション。
- **コントロールノード** (このシステムから Red Hat Ansible Core は他のシステムを設定) へのアクセスおよびパーミッション。
  コントロールノードでは、
  - **ansible-core** パッケージおよび **rhel-system-roles** パッケージがインストールされている。

重要

RHEL 8.0-8.5 では、別の Ansible リポジトリーへのアクセス権を指定されており、Ansible をベースにする自動化用の Ansible Engine 2.9 が含まれています。Ansible Engine には、**ansible**、**ansible-playbook** などのコマンドラインユーティリティー、**docker** や **podman** などのコネクター、プラグインとモジュールが多く含まれています。Ansible Engine を入手してインストールする方法の詳細は、ナレッジベースのアーティクル記事 How to download and install Red Hat Ansible Engine を参照してください。

RHEL 8.6 および 9.0 では、Ansible Core (**ansible-core** パッケージとして提供) が導入されました。これには、Ansible コマンドラインユーティリティー、コマンド、およびビルトイン Ansible プラグインのセットが含まれています。RHEL は、AppStream リポジトリーを介してこのパッケージを提供し、サポート範囲は限定的です。詳細については、ナレッジベースの Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories を参照してください。

- マネージドノードが記載されているインベントリーファイルがある。

手順

1. **sshd** システムロールの Playbook の例をコピーします。

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

2. 以下の例のように、テキストエディターでコピーした Playbook を開きます。

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

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

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

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

設定に合わせて変数を変更できます。詳細は、SSH サーバーのシステムロール変数 を参照してください。

3. オプション: Playbook の構文を確認します。

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

4. インベントリーファイルで Playbook を実行します。

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

...

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

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

検証

1. SSH サーバーにログインします。

```
$ ssh user1@10.1.1.1
```

詳細は以下のようになります。

- **user1** は、SSH サーバーのユーザーです。
- **10.1.1.1** は、SSH サーバーの IP アドレスです。

2. SSH サーバーの **sshd_config** ファイルの内容を確認します。

```
$ cat /etc/ssh/sshd_config
…
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@10.1.1.1
```

関連情報

- **/usr/share/doc/rhel-system-roles/sshd/README.md** ファイル。
- **ansible-playbook(1)** の man ページ。

14.3. SSH システムロール変数

ssh システムロール Playbook では、設定および制限に応じて、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。

14.4. SSH システムロールを使用した OPENSSH クライアントの設定

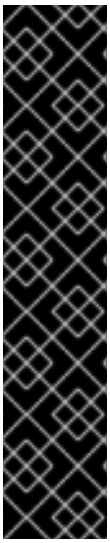
ssh システムロールを使用して、Ansible Playbook を実行して複数の SSH クライアントを設定できます。

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 (RHEL 8 から デフォルト) を使用していることを確認してください。

前提条件

- 1 つ以上の **管理対象ノード** (**ssh** システムロールで設定するシステム) へのアクセスおよびパーミッション。
- **コントロールノード** (このシステムから Red Hat Ansible Core は他のシステムを設定) へのアクセスおよびパーミッション。
  コントロールノードでは、
  - **ansible-core** パッケージおよび **rhel-system-roles** パッケージがインストールされている。



重要

RHEL 8.0-8.5 では、別の Ansible リポジトリーへのアクセス権を指定されており、Ansible をベースにする自動化用の Ansible Engine 2.9 が含まれています。Ansible Engine には、**ansible**、**ansible-playbook** などのコマンドラインユーティリティー、**docker** や **podman** などのコネクター、プラグインとモジュールが多く含まれています。Ansible Engine を入手してインストールする方法の詳細は、ナレッジベースのアーティクル記事 How to download and install Red Hat Ansible Engine を参照してください。

RHEL 8.6 および 9.0 では、Ansible Core (**ansible-core** パッケージとして提供) が導入されました。これには、Ansible コマンドラインユーティリティー、コマンド、およびビルトイン Ansible プラグインのセットが含まれています。RHEL は、AppStream リポジトリーを介してこのパッケージを提供し、サポート範囲は限定的です。詳細については、ナレッジベースの Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories を参照してください。

- マネージドノードが記載されているインベントリーファイルがある。

手順

1. 以下の内容を含む新しい **playbook.yml** ファイルを作成します。

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

```
      - Condition: example
        Hostname: example.com
        User: user1
    ssh_ForwardX11: no
```

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。
- **example.com** ホストに接続するための **example** エイリアスが **user1** である。
- ホストエイリアスの **example** が作成されている。(これはユーザー名が **user1** の **example.com** ホストへの接続を表します。)
- X11 転送が無効化されている。

必要に応じて、これらの変数は設定に合わせて変更できます。詳細は、**ssh** System Role variables を参照してください。

2. オプション: Playbook の構文を確認します。

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

3. インベントリーファイルで Playbook を実行します。

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

検証

- テキストエディターで SSH 設定ファイルを開いて、マネージドノードが正しく設定されていることを確認します。以下に例を示します。

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

上記の Playbook の例の適用後に、設定ファイルの内容は以下のようになるはずです。

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

14.5. 非排他的な設定のための SSHD システムロールの使用

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が発生する可能性があります。他のオプションをそのまま維持しながら、選択した設定オプションのみに **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。

RHEL 8 以前では、設定スニペットを使用して非排他的設定を適用することができます。

前提条件

- 1 つ以上の **管理対象ノード** (**sshd** システムロールで設定するシステム) へのアクセスおよびパーミッション。
- **コントロールノード** (このシステムから Red Hat Ansible Core は他のシステムを設定) へのアクセスおよびパーミッション。
  コントロールノードでは、
  - **ansible-core** パッケージがインストールされている。
  - マネージドノードが記載されているインベントリーファイルがある。
  - 別の RHEL システムロールの Playbook。

手順

1. **sshd_config_namespace** 変数を含む設定スニペットを Playbook に追加します。

```
---
- hosts: all
  tasks:
  - name: <Configure SSHD to accept some useful environment variables>
    include_role:
      name: rhel-system-roles.sshd
    vars:
      sshd_config_namespace: <my-application>
      sshd:
        # Environment variables to accept
        AcceptEnv:
          LANG
          LS_COLORS
          EDITOR
```

Playbook をインベントリーに適用すると、ロールは次のスニペットがない場合は **/etc/ssh/sshd_config** ファイルに追加します。

```
# 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>
```

検証

- オプション: Playbook の構文を確認します。

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

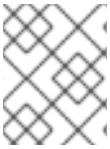
関連情報

- **/usr/share/doc/rhel-system-roles/sshd/README.md** ファイル。
- **ansible-playbook(1)** の man ページ。

第15章 VPN RHEL システムロールを使用した IPSEC との 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。

15.1. VPN システムロールを使用して IPSEC でホスト間 VPN の作成

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

- 1 つ以上の **管理対象ノード** (**vpn** システムロールで設定するシステム) へのアクセスおよびパーミッション。
- **コントロールノード** (このシステムから Red Hat Ansible Core は他のシステムを設定) へのアクセスおよびパーミッション。
  コントロールノードでは、
  - **ansible-core** パッケージおよび **rhel-system-roles** パッケージがインストールされている。

重要

RHEL 8.0-8.5 では、別の Ansible リポジトリーへのアクセス権を指定されており、Ansible をベースにする自動化用の Ansible Engine 2.9 が含まれています。Ansible Engine には、**ansible**、**ansible-playbook** などのコマンドラインユーティリティー、**docker** や **podman** などのコネクター、プラグインとモジュールが多く含まれています。Ansible Engine を入手してインストールする方法の詳細は、ナレッジベースのアーティクル記事 How to download and install Red Hat Ansible Engine を参照してください。

RHEL 8.6 および 9.0 では、Ansible Core (**ansible-core** パッケージとして提供) が導入されました。これには、Ansible コマンドラインユーティリティー、コマンド、およびビルトイン Ansible プラグインのセットが含まれています。RHEL は、AppStream リポジトリーを介してこのパッケージを提供し、サポート範囲は限定的です。詳細については、ナレッジベースの Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories を参照してください。

- マネージドノードが記載されているインベントリーファイルがある。

手順

1. 以下の内容を含む新しい **playbook.yml** ファイルを作成します。

```
- name: Host to host VPN
  hosts: managed_node1, managed_node2
  roles:
    - rhel-system-roles.vpn
  vars:
    vpn_connections:
      - hosts:
          managed_node1:
          managed_node2:
    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、**managed_node1からmanaged_node2** への接続を設定します。**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** 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. 必要に応じて、ホストの **vpn_connections** リストに次のセクションを追加して、マネージドホストから、インベントリーファイルに記述されていない外部ホストへの接続を設定します。

```
    vpn_connections:
      - hosts:
          managed_node1:
          managed_node2:
          external_node:
            hostname: 192.0.2.2
```

これは、追加の接続 (**managed_node1からexternal_node**) へと (**managed_node2からexternal_node**) を設定します。

注記

接続はマネージドノードでのみ設定され、外部ノードでは設定されません。

1. 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、マネージドノードに複数の VPN 接続を指定できます。

```
- name: Multiple VPN
  hosts: managed_node1, managed_node2
  roles:
    - rhel-system-roles.vpn
  vars:
    vpn_connections:
      - name: control_plane_vpn
        hosts:
          managed_node1:
            hostname: 192.0.2.0 # IP for the control plane
          managed_node2:
            hostname: 192.0.2.1
      - name: data_plane_vpn
```

```
    hosts:
      managed_node1:
        hostname: 10.0.0.1 # IP for the data plane
      managed_node2:
        hostname: 10.0.0.2
```

2. 必要に応じて、設定に合わせて変数を変更できます。詳細は、**/usr/share/doc/rhel-system-roles/vpn/README.md** ファイルを参照してください。

3. オプション: Playbook の構文を確認します。

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

4. インベントリーファイルで Playbook を実行します。

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

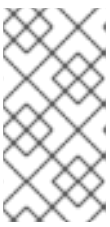
デフォルトでは、ロールは、各システムの観点から作成する接続ごとにわかりやすい名前を生成します。たとえば、**managed_node1** と **managed_node2** との間の接続を作成するときに、**managed_node1** 上のこの接続のわかりやすい名前は **managed_node1-to-managed_node2** ですが、**managed_node2** では、この接続の名前は **managed_node2-to-managed_node1** となります。

1. 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
```

2. 必要に応じて、接続が正常に読み込まれなかった場合は、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

接続の読み込みおよび開始のプロセス中に発生した可能性のあるエラーは、ログに報告されます。ログは、**/var/log/pluto.log** にあります。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みてください。

15.2. VPN システムロールを使用して IPSEC でオポチュニスティックメッシュ VPN 接続の作成

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。

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。**vpn** システムロールは、**/etc/ipsec.d** ディレクトリーで定義されている 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ことを前提としています。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、これは **managed_node1** 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。

次の手順例では、Ansible Playbook を実行するシステムであるコントロールノードは、両方のマネージドノード (192.0.2.0/24) と同じクラスレスドメイン間ルーティング (CIDR) 番号を共有し、IP アドレスは 192.0.2.7 になります。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。

前提条件

- 1 つ以上の **管理対象ノード** (**vpn** システムロールで設定するシステム) へのアクセスおよびパーミッション。
  - すべてのマネージドノードで、**/etc/ipsec.d** ディレクトリーの NSS データベースには、ピア認証に必要なすべての証明書が含まれています。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。
- **コントロールノード** (このシステムから Red Hat Ansible Core は他のシステムを設定) へのアクセスおよびパーミッション。
  コントロールノードでは、
  - **ansible-core** パッケージおよび **rhel-system-roles** パッケージがインストールされている。

重要

RHEL 8.0-8.5 では、別の Ansible リポジトリーへのアクセス権を指定されており、Ansible をベースにする自動化用の Ansible Engine 2.9 が含まれています。Ansible Engine には、**ansible**、**ansible-playbook** などのコマンドラインユーティリティー、**docker** や **podman** などのコネクター、プラグインとモジュールが多く含まれています。Ansible Engine を入手してインストールする方法の詳細は、ナレッジベースのアーティクル記事 How to download and install Red Hat Ansible Engine を参照してください。

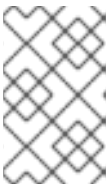
RHEL 8.6 および 9.0 では、Ansible Core (**ansible-core** パッケージとして提供) が導入されました。これには、Ansible コマンドラインユーティリティー、コマンド、およびビルトイン Ansible プラグインのセットが含まれています。RHEL は、AppStream リポジトリーを介してこのパッケージを提供し、サポート範囲は限定的です。詳細については、ナレッジベースの Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories を参照してください。

- マネージドノードが記載されているインベントリーファイルがある。

手順

1. 以下の内容を含む新しい **playbook.yml** ファイルを作成します。

```
- name: Mesh VPN
  hosts: managed_node1, managed_node2, managed_node3
  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
```



注記

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** 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. 必要に応じて、設定に合わせて変数を変更できます。詳細は、**/usr/share/doc/rhel-system-roles/vpn/README.md** ファイルを参照してください。

3. オプション: Playbook の構文を確認します。

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

4. インベントリーファイルで Playbook を実行します。

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

15.3. 関連情報

- VPN システムロールで使用するパラメーターの詳細と、**vpn** システムロールに関する追加情報は、**/usr/share/doc/rhel-system-roles/vpn/README.md** ファイルを参照してください。
- **ansible-playbook** コマンドの詳細は、man ページの **ansible-playbook(1)** を参照してください。

第16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、多くの異なるシステムで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。

16.1. CRYPTO_POLICIES システムロール変数およびファクト

C**crypto_policies** システムロール Playbook では、設定および制限に合わせて、**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

管理対象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

- 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

16.2. 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

- **crypto_policies** システムロールで設定するシステムである 1 つ以上の **管理対象ノード** へのアクセスとパーミッション。

- コントロールノード (このシステムから Red Hat Ansible Core は他のシステムを設定) へのアクセスおよびパーミッション。
  コントロールノードでは、
  - **ansible-core** パッケージおよび **rhel-system-roles** パッケージがインストールされている。

重要

RHEL 8.0-8.5 では、別の Ansible リポジトリーへのアクセス権を指定されており、Ansible をベースにする自動化用の Ansible Engine 2.9 が含まれています。Ansible Engine には、**ansible**、**ansible-playbook** などのコマンドラインユーティリティー、**docker** や **podman** などのコネクター、プラグインとモジュールが多く含まれています。Ansible Engine を入手してインストールする方法の詳細は、ナレッジベースのアーティクル記事 How to download and install Red Hat Ansible Engine を参照してください。

RHEL 8.6 および 9.0 では、Ansible Core (**ansible-core** パッケージとして提供) が導入されました。これには、Ansible コマンドラインユーティリティー、コマンド、およびビルトイン Ansible プラグインのセットが含まれています。RHEL は、AppStream リポジトリーを介してこのパッケージを提供し、サポート範囲は限定的です。詳細については、ナレッジベースの Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories を参照してください。

- マネージドノードが記載されているインベントリーファイルがある。

手順

1. 以下の内容を含む新しい **playbook.yml** ファイルを作成します。

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

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 変数を設定すると、システムロールで暗号化ポリシーを変更した後にシステムが再起動されます。

詳細については、crypto_policies システムロールの変数とファクト を参照してください。

2. オプション: Playbook の構文を確認します。

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

3. インベントリーファイルで Playbook を実行します。

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

検証

1. コントロールノードで (例: **verify_playbook.yml**) という名前の別の Playbook を作成します。

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

 - debug:
     var: crypto_policies_active
```

この Playbook では、システムの設定は変更されず、マネージドノードのアクティブなポリシーだけを報告します。

2. 同じインベントリーファイルで Playbook を実行します。

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

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

"crypto_policies_active": 変数は、マネージドノードでアクティブなポリシーを表示します。

16.3. 関連情報

- **/usr/share/ansible/roles/rhel-system-roles.crypto_policies/README.md** ファイル
- **ansible-playbook(1)** の man ページ。
- RHEL System Roles を使用するための制御ノードと管理対象ノードの準備

第17章 RHEL システムロールを使用した NBDE の設定

17.1. 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集合です。

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

- NBDE (Network-Bound Disk Encryption) ロール変数の詳細は、**rhel-system-roles** パッケージをインストールし、**/usr/share/doc/rhel-system-roles/nbde_client/** と **/usr/share/doc/rhel-system-roles/nbde_server/** ディレクトリーの **README.md** と **README.html** ファイルを参照してください。
- たとえば、system-roles Playbook の場合は、**rhel-system-roles** パッケージをインストールし、**/usr/share/ansible/roles/rhel-system-roles.nbde_server/examples/** ディレクトリーを参照してください。
- RHEL システムロールの詳細は、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 を参照してください。

17.2. 複数の TANG サーバーをセットアップするための NBDE_SERVER システムロールの使用

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

- 1 つ以上の **マネージドノード** (**nbde_server** システムロールで設定するシステム) へのアクセスおよびパーミッション。
- コントロールノード (このシステムから Red Hat Ansible Core は他のシステムを設定) へのアクセスおよびパーミッション。
  コントロールノードでは、
  - **ansible-core** パッケージおよび **rhel-system-roles** パッケージがインストールされている。

重要

RHEL 8.0-8.5 では、別の Ansible リポジトリーへのアクセス権を指定されており、Ansible をベースにする自動化用の Ansible Engine 2.9 が含まれています。Ansible Engine には、**ansible**、**ansible-playbook** などのコマンドラインユーティリティー、**docker** や **podman** などのコネクター、プラグインとモジュールが多く含まれています。Ansible Engine を入手してインストールする方法の詳細は、ナレッジベースのアーティクル記事 How to download and install Red Hat Ansible Engine を参照してください。

RHEL 8.6 および 9.0 では、Ansible Core (**ansible-core** パッケージとして提供) が導入されました。これには、Ansible コマンドラインユーティリティー、コマンド、およびビルトイン Ansible プラグインのセットが含まれています。RHEL は、AppStream リポジトリーを介してこのパッケージを提供し、サポート範囲は限定的です。詳細については、ナレッジベースの Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories を参照してください。

- マネージドノードが記載されているインベントリーファイルがある。

手順

1. Tang サーバーの設定が含まれる Playbook を準備します。ゼロから開始するか、**/usr/share/ansible/roles/rhel-system-roles.nbde_server/examples/** ディレクトリーにある Playbook のいずれかのサンプルを使用することができます。

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

2. 選択したテキストエディターで Playbook を編集します。以下に例を示します。

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

3. 必要なパラメーターを追加します。以下の Playbook の例では、Tang サーバーのデプロイと鍵のローテーションを確実に実行します。

```
---
- hosts: all
```

```
vars:
  nbde_server_rotate_keys: yes
  nbde_server_manage_firewall: true
  nbde_server_manage_selinux: true

roles:
  - rhel-system-roles.nbde_server
```

注記

nbde_server_manage_firewall と **nbde_server_manage_selinux** は両方とも true に設定されているため、**nbde_server** ロールは **firewall** 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。

4. 終了した Playbook を適用します。

```
# ansible-playbook -i inventory-file my-tang-playbook.yml
```

ここで、* **inventory-file** はインベントリーファイルに置き換えます。* **logging-playbook.yml** は、使用する Playbook に置き換えます。

重要

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

- 詳細は、**rhel-system-roles** パッケージをインストールして、**/usr/share/doc/rhel-system-roles/nbde_server/** ディレクトリーおよび **usr/share/ansible/roles/rhel-system-roles.nbde_server/** ディレクトリーを参照してください。

17.3. 複数の CLEVIS クライアントの設定に NBDE_CLIENT システムロールを使用

手順に従って、Clevis クライアント設定を含む Ansible Playbook を準備および適用します。

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。これは、現時点では TPM2 バインディングに使用できないことを意味します。

前提条件

- 1 つ以上の **マネージドノード** (**nbde_client** システムロールで設定するシステム) へのアクセスおよびパーミッション。

- **コントロールノード** (このシステムから Red Hat Ansible Core は他のシステムを設定) へのアクセスおよびパーミッション。
- Ansible Core パッケージがコントロールマシンにインストールされている。
- **rhel-system-roles** パッケージが、Playbook を実行するシステムにインストールされている。

手順

1. Clevis クライアントの設定が含まれる Playbook を準備します。ゼロから開始するか、**/usr/share/ansible/roles/rhel-system-roles.nbde_client/examples/** ディレクトリーにある Playbook のいずれかのサンプルを使用することができます。

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

2. 選択したテキストエディターで Playbook を編集します。以下に例を示します。

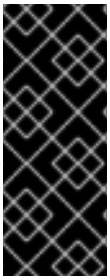
```
# vi my-clevis-playbook.yml
```

3. 必要なパラメーターを追加します。以下の Playbook の例では、2 つの 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。

```
---
- hosts: all

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

  roles:
    - rhel-system-roles.nbde_client
```

4. 終了した Playbook を適用します。

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



重要

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

- パラメーターの詳細と、NBDE Client システムロールに関する追加情報は、**rhel-system-roles** パッケージをインストールし、**/usr/share/doc/rhel-system-roles/nbde_client/** および **/usr/share/ansible/roles/rhel-system-roles.nbde_client/** ディレクトリーを参照してください。

第18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、Red Hat Ansible Core を使用して証明書の発行および管理ができます。

本章では、以下のトピックについて説明します。

- **certificate** システムロール
- **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した IdM CA からの新しい証明書の要求

18.1. CERTIFICATE システムロール

certificate システムロールを使用して、Ansible Core を使用し、TLS および SSL の証明書の発行および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 システムロールを使用すると、Ansible Playbook で以下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** ファイルを参照してください。
- RHEL System Roles を使用するための制御ノードと管理対象ノードの準備

18.2. 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 システムロールでは、Ansible Core を使用して自己署名の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。

注記

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。

前提条件

- Ansible Core パッケージがコントロールマシンにインストールされている。
- Playbook を実行するシステムに **rhel-system-roles** パッケージがインストールされている。

手順

手順

1. **オプション: inventory.file** などのインベントリーファイルを作成します。

```
$ *touch inventory.file*
```

2. インベントリーファイルを開き、証明書を要求するホストを定義します。以下に例を示します。

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

3. Playbook ファイルを作成します (例: **request-certificate.yml**)。
   - **webserver** など、証明書を要求するホストを含むように **hosts** を設定します。
   - **certificate_requests** 変数を設定して以下を追加します。
     - **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。
   - **roles** の下に **rhel-system-roles.certificate** ロールを設定します。
     以下は、この例の Playbook ファイルです。

```
---
- hosts: webserver

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

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

4. ファイルを保存します。

5. Playbook を実行します。

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

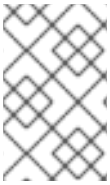
関連情報

- **/usr/share/ansible/roles/rhel-system-roles.certificate/README.md** ファイルを参照してください。
- **ansible-playbook(1)** man ページを参照してください。

18.3. 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 システムロールでは、統合認証局 (CA) で IdM サーバーを使用しているときに、 **anible-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。



注記

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。

前提条件

- Ansible Core パッケージがコントロールマシンにインストールされている。
- Playbook を実行するシステムに **rhel-system-roles** パッケージがインストールされている。

手順

1. **オプション: inventory.file** などのインベントリーファイルを作成します。

```
$ *touch inventory.file*
```

2. インベントリーファイルを開き、証明書を要求するホストを定義します。以下に例を示します。

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

3. Playbook ファイルを作成します (例: **request-certificate.yml**)。
   - **webserver** など、証明書を要求するホストを含むように **hosts** を設定します。
   - **certificate_requests** 変数を設定して以下を追加します。
     - **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。
   - **roles** の下に **rhel-system-roles.certificate** ロールを設定します。
     以下は、この例の Playbook ファイルです。

```
---
- hosts: webserver
  vars:
    certificate_requests:
      - name: mycert
        dns: www.example.com
```

```
      principal: HTTP/www.example.com@EXAMPLE.COM
      ca: ipa

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

4. ファイルを保存します。
5. Playbook を実行します。

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

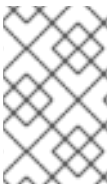
関連情報

- **/usr/share/ansible/roles/rhel-system-roles.certificate/README.md** ファイルを参照してください。
- **ansible-playbook(1)** man ページを参照してください。

18.4. CERTIFICATE システムロールを使用して、証明書の発行前または発行後に実行するコマンドの指定

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。



注記

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。

前提条件

- Ansible Core パッケージがコントロールマシンにインストールされている。
- Playbook を実行するシステムに **rhel-system-roles** パッケージがインストールされている。

手順

1. **オプション: inventory.file** などのインベントリーファイルを作成します。

```
$ *touch inventory.file*
```

2. インベントリーファイルを開き、証明書を要求するホストを定義します。以下に例を示します。

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

3. Playbook ファイルを作成します (例: **request-certificate.yml**)。

- **webserver** など、証明書を要求するホストを含むように **hosts** を設定します。
- **certificate_requests** 変数を設定して以下を追加します。
  - **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**)。
- **roles** の下に **rhel-system-roles.certificate** ロールを設定します。
  以下は、この例の Playbook ファイルです。

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

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

4. ファイルを保存します。
5. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.certificate/README.md** ファイルを参照してください。
- **ansible-playbook(1)** man ページを参照してください。

第19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 7.9 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。

19.1. KDUMP RHEL システムロール

kdump システムロールを使用すると、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。

19.2. KDUMP ロールのパラメーター

kdump 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
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と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。

関連情報

- makedumpfile(8) の man ページ
- **kdump** で使用されるパラメーターの詳細および **kdump** システムロールに関する追加情報は、**/usr/share/ansible/roles/rhel-system-roles.tlog/README.md** ファイルを参照してください。

19.3. RHEL システムロールを使用した KDUMP の設定

Ansible Playbook を実行して **kdump** システムロールを使用し、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 ロールは、**/etc/kdump.conf** ファイルを置き換えることで、マネージドホストの kdump 設定全体を置き換えます。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

- Ansible Core パッケージがコントロールマシンにインストールされている。
- Playbook を実行するシステムに **rhel-system-roles** パッケージがインストールされている。
- **kdump** をデプロイするシステムをリスト表示するインベントリーファイルがある。

手順

1. 以下の内容を含む新しい **playbook.yml** ファイルを作成します。

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

2. オプション: Playbook の構文を確認します。

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

3. インベントリーファイルで Playbook を実行します。

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

関連情報

- **kdump** ロール変数の詳細は、/usr/share/doc/rhel-system-roles/kdump ディレクトリーの README.md ファイルまたは README.html ファイルを参照してください。
- Preparing the control node and managed nodes to use RHEL System Roles を参照してください。
- **rhel-system-roles** パッケージをインストールし、**/usr/share/ansible/roles/rhel-system-roles.kdump/README.html** のドキュメントを参照してください。

第20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 8 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。

RHEL システムロールと、その適用方法の詳細は、RHEL システムロールの概要 を参照してください。

20.1. 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

20.2. STORAGE システムロールで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、以下の変数に記載されているファイルシステム、ボリューム、およびプールにのみ影響します。

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。

20.3. 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

本セクションでは、Ansible Playbook の例を紹介します。この Playbook では、**storage** ロールを適用し、デフォルトパラメーター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します。

警告

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。

例20.1 /dev/sdb に XFS を作成する Playbook

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

- 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 8 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、Example Ansible playbook to manage logical volumes 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。

関連情報

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

20.4. ファイルシステムを永続的にマウントする ANSIBLE PLAYBOOK の例

本セクションでは、Ansible Playbook の例を紹介します。この Playbook は、**storage** ロールをすぐに適用して、XFS ファイルシステムを永続的にマウントします。

例20.2 /dev/sdb のファイルシステムを /mnt/data にマウントする Playbook

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

- この Playbook では、ファイルシステムが **/etc/fstab** ファイルに追加され、すぐにファイルシステムをマウントします。
- **/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。

関連情報

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

20.5. 論理ボリュームを管理する ANSIBLE PLAYBOOK の例

本セクションでは、Ansible Playbook の例を紹介します。この Playbook 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。

例20.3 myvg ボリュームグループに mylv 論理ボリュームを作成する Playbook

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

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

- **myvg** ボリュームグループは、次のディスクで設定されます。
  - **/dev/sda**
  - **/dev/sdb**
  - **/dev/sdc**
- **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。
- Playbook は、**mylv** 論理ボリューム上に Ext4 ファイルシステムを作成し、**/mnt** ファイルシステムを永続的にマウントします。

関連情報

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

20.6. オンラインのブロック破棄を有効にする ANSIBLE PLAYBOOK の例

本セクションでは、Ansible Playbook の例を紹介します。この Playbook 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。

例20.4 /mnt/data/ でのオンラインのブロック破棄を有効にする Playbook

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

関連情報

- ファイルシステムを永続的にマウントする Ansible Playbook の例
- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** ファイル

20.7. EXT4 ファイルシステムを作成してマウントする ANSIBLE PLAYBOOK の例

本セクションでは、Ansible Playbook の例を紹介します。この Playbook 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。

例20.5 /dev/sdb に Ext4 を作成し、/mnt/data にマウントする Playbook

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

- Playbook は、**/dev/sdb** ディスクにファイルシステムを作成します。
- Playbook は、**/mnt/data** ディレクトリーにファイルシステムを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。

関連情報

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

20.8. EXT3 ファイルシステムを作成してマウントする ANSIBLE PLAYBOOK の例

本セクションでは、Ansible Playbook の例を紹介します。この Playbook 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。

例20.6 /dev/sdb に Ext3 を作成し、/mnt/data にマウントする Playbook

```
---
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
```

```
      disks:
        - sdb
      fs_type: ext3
      fs_label: label-name
      mount_point: /mnt/data
  roles:
    - rhel-system-roles.storage
```

- Playbook は、**/dev/sdb** ディスクにファイルシステムを作成します。
- Playbook は、**/mnt/data** ディレクトリーにファイルシステムを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。

関連情報

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

20.9. STORAGE RHEL システムロールを使用して既存の EXT4 または EXT3 ファイルシステムのサイズを変更する ANSIBLE PLAYBOOK の例

本セクションでは、Ansible Playbook の例を紹介します。この Playbook は、**storage** ロールを適用して、ブロックデバイスにある既存の Ext4 または Ext3 ファイルシステムのサイズを変更します。

例20.7 ディスクに単一ボリュームを設定する Playbook

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

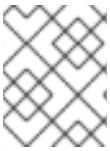
- 直前の例のボリュームがすでに存在する場合に、ボリュームサイズを変更するには、異なるパラメーター **size** の値で、同じ Playbook を実行する必要があります。以下に例を示します。

例20.8 /dev/sdb で ext4 のサイズを変更する Playbook

```
---
- name: Create a disk device mounted on /opt/barefs
- hosts: all
  vars:
    storage_volumes:
```

```
    - name: barefs
      type: disk
      disks:
        - /dev/sdb
      size: 10 GiB
      fs_type: ext4
      mount_point: /opt/barefs
roles:
  - rhel-system-roles.storage
```

- 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。



注記

他のファイルシステムで **Resizing** アクションを使用すると、作業しているデバイスのデータを破棄する可能性があります。

関連情報

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

20.10. 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 ANSIBLE PLAYBOOK の例

本セクションでは、Ansible Playbook の例を紹介します。この 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使用して LVM 論理ボリュームのサイズを変更します。

警告

他のファイルシステムで **Resizing** アクションを使用すると、作業しているデバイスのデータを破棄する可能性があります。

例20.9 myvg ボリュームグループの既存の mylv1 および myvl2 論理ボリュームのサイズを変更する Playbook

```
---

- hosts: all
  vars:
    storage_pools:
      - name: myvg
        disks:
          - /dev/sda
          - /dev/sdb
          - /dev/sdc
        volumes:
```

```
        - name: mylv1
          size: 10 GiB
          fs_type: ext4
          mount_point: /opt/mount1
        - name: mylv2
          size: 50 GiB
          fs_type: ext4
          mount_point: /opt/mount2

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

- 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。

関連情報

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

20.11. STORAGE RHEL SYSTEM ROLE を使用してスワップボリュームを作成する ANSIBLE PLAYBOOK の例

本セクションでは、Ansible Playbook の例を紹介します。この Playbook は、**storage** ロールを適用して、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、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変更します。

例20.10 /dev/sdb で既存の XFS を作成または変更する Playbook

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

- 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。

関連情報

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

20.12. STORAGE システムロールを使用した RAID ボリュームの設定

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。

前提条件

- Ansible Core パッケージがコントロールマシンにインストールされている。
- Playbook を実行するシステムに **rhel-system-roles** パッケージがインストールされている。
- **storage** システムロールを使用して、RAID ボリュームをデプロイするシステムの詳細を記録したインベントリーファイルがある。

手順

1. 以下のコンテンツを含む新しい **playbook.yml** ファイルを作成します。

```
---
- name: Configure the storage
  hosts: managed-node-01.example.com
  tasks:
  - name: Create a RAID on sdd, sde, sdf, and sdg
    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
```

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。

2. オプション: Playbook の構文を確認します。

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

3. Playbook を実行します。

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

関連情報

- RAID の管理
- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** ファイル
- 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

20.13. STORAGE RHEL SYSTEM ROLE を使用して RAID で 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 RHEL に LVM pool with RAID を設定できます。本セクションでは、利用可能なパラメーターを使用して Ansible Playbook を設定し、LVM pool with RAID を設定する方法を説明します。

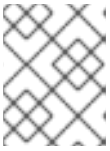
前提条件

- Ansible Core パッケージがコントロールマシンにインストールされている。
- Playbook を実行するシステムに **rhel-system-roles** パッケージがインストールされている。
- **storage** システムロールを使用して、LVM pool with RAID を設定するシステムの詳細を記録したインベントリーファイルがある。

手順

1. 以下の内容を含む新しい **playbook.yml** ファイルを作成します。

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



注記

LVM pool with RAID 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. オプション:Playbook の構文を確認します。

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

3. インベントリーファイルで Playbook を実行します。

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

関連情報

- RAID の管理
- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** ファイル

20.14. STORAGE RHEL システムロールを使用し、LVM 上の VDO ボリュームを圧縮および重複排除する ANSIBLE PLAYBOOK の例

本セクションでは、Ansible Playbook の例を紹介します。この Playbook は **storage** RHEL System Role を適用して、Virtual Data Optimizer (VDO) を使用して論理ボリューム (LVM) の圧縮および重複排除を有効にします。

例20.11 myvg ボリュームグループに、mylv1 LVM VDO ボリュームを作成する Playbook

```
---
- name: Create LVM VDO volume under volume group 'myvg'
  hosts: all
  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。

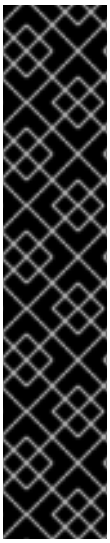
- 圧縮は、ストレージボリュームに保存されているデータを圧縮するために使用されます。これにより、より大きなストレージ容量が得られます。
- vdo_pool_size は、ボリュームがデバイスで使用する実際のサイズを指定します。VDO ボリュームの仮想サイズは、**size** パラメーターで設定します。注記: LVM VDO はストレージロールで使用されるため、プールごとに圧縮と重複排除を使用できるボリュームは 1 つだけです。

20.15. STORAGE RHEL システムロールを使用した LUKS2 暗号化ボリュームの作成

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

- **crypto_policies** システムロールで設定するシステムである 1 つ以上の管理対象ノードへのアクセスとパーミッションがある。
- 管理対象ノードが記載されているインベントリーファイルがある。
- コントロールノード (このシステムから Red Hat Ansible Core は他のシステムを設定) へのアクセスおよびパーミッション。**ansible-core** パッケージおよび **rhel-system-roles** パッケージがコントロールノードにインストールされている。



重要

RHEL 8.0-8.5 では、別の Ansible リポジトリーへのアクセス権を指定されており、Ansible をベースにする自動化用の Ansible Engine 2.9 が含まれています。Ansible Engine には、**ansible**、**ansible-playbook** などのコマンドラインユーティリティー、**docker** や **podman** などのコネクター、プラグインとモジュールが多く含まれています。Ansible Engine を入手してインストールする方法の詳細は、ナレッジベースのアーティクル記事 How to download and install Red Hat Ansible Engine を参照してください。

RHEL 8.6 および 9.0 では、Ansible Core (**ansible-core** パッケージとして提供) が導入されました。これには、Ansible コマンドラインユーティリティー、コマンド、およびビルトイン Ansible プラグインのセットが含まれています。RHEL は、AppStream リポジトリーを介してこのパッケージを提供し、サポート範囲は限定的です。詳細については、ナレッジベースの Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories を参照してください。

手順

1. 以下の内容を含む新しい **playbook.yml** ファイルを作成します。

```
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
         - sdb
        fs_type: xfs
        fs_label: label-name
        mount_point: /mnt/data
```

```
      encryption: true
      encryption_password: your-password
  roles:
   - rhel-system-roles.storage
```

playbook.yml ファイルに、**encryption_key**、**encryption_cipher**、**encryption_key_size**、および **encryption_luks** バージョンなどの他の暗号化パラメーターを追加することもできます。

2. オプション: Playbook の構文を確認します。

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

3. インベントリーファイルで Playbook を実行します。

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

3. **storage** ロールがサポートする **playbook.yml** ファイル内の **cryptsetup** パラメーターを表示します。

```
# cat ~/playbook.yml

    - hosts: all
      vars:
        storage_volumes:
          - name: foo
            type: disk
            disks:
             - nvme0n1
            fs_type: xfs
            fs_label: label-name
            mount_point: /mnt/data
            encryption: true
            #encryption_password: passwdpasswd
            encryption_key: /home/passwd_key
            encryption_cipher: aes-xts-plain64
            encryption_key_size: 512
            encryption_luks_version: luks2

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

関連情報

- LUKS を使用したブロックデバイスの暗号化
- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file

20.16. STORAGE RHEL システムロールを使用し、プールのボリュームサイズをパーセンテージで表す ANSIBLE PLAYBOOK の例

本セクションでは、Ansible Playbook の例を紹介します。この Playbook では、**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) ボリュームの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。

例20.12 ボリュームのサイズをプールの合計サイズのパーセンテージで表現する Playbook

```
---
- name: Express volume sizes as a percentage of the pool's total size
  hosts: all
  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%")。また、LVM ボリュームのサイズを、人間が判読できるファイルシステムのサイズ ("10g" や "50 GiB" など) で、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。

20.17. 関連情報

- **/usr/share/doc/rhel-system-roles/storage/**
- **/usr/share/ansible/roles/rhel-system-roles.storage/**

第21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。

21.1. 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** または **chrony** を使用して NTP プロトコルを実装するかどうかにかかわらず、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。

21.2. サーバーの 1 つのプールへの TIMESYNC システムロールの適用

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

- Ansible Core パッケージがコントロールマシンにインストールされている。
- Playbook を実行するシステムに **rhel-system-roles** パッケージがインストールされている。
- **timesync** システムロールをデプロイするシステムをリスト表示するインベントリーファイルがある。

手順

1. 以下の内容を含む新しい **playbook.yml** ファイルを作成します。

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

```
      iburst: yes
  roles:
    - rhel-system-roles.timesync
```

2. オプション: Playbook の構文を確認します。

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

3. インベントリーファイルで Playbook を実行します。

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

21.3. クライアントサーバーでの TIMESYNC システムロールの適用

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

- **timesync** ソリューションをデプロイするシステムに Red Hat Ansible Automation Platform をインストールする必要はありません。
- Playbook を実行するシステムに **rhel-system-roles** パッケージがインストールされている。
- **timesync** システムロールをデプロイするシステムのリストを示すインベントリーファイルがある。
- **chrony** の NTP プロバイダーバージョンは 4.0 以降。

手順

1. 以下の内容を含む **playbook.yml** ファイルを作成します。

```
---
- hosts: timesync-test
  vars:
    timesync_ntp_servers:
      - hostname: ptbtime1.ptb.de
        iburst: yes
```

```
      nts: yes
  roles:
    - rhel-system-roles.timesync
```

ptbtime1.ptb.de は、公開サーバーの例です。別のパブリックサーバーまたは独自のサーバーを使用できます。

2. オプション: Playbook の構文を確認します。

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

3. インベントリーファイルで Playbook を実行します。

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

- **chrony.conf(5)** の man ページ

21.4. TIMESYNC システムロール変数

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

ロール変数の設定	説明
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

- **timesync** ロール変数の詳細は、rhel-system-roles パッケージをインストールし、**/usr/share/doc/rhel-system-roles/timesync** ディレクトリーの README.md または README.html ファイルを参照してください。

第22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、Ansible Automation Platform コントロールノードで **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。

22.1. 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集合です。**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。これには、オプションでローカルシステムによって監視されるリモートシステムの一覧が含まれます。**metrics** システムロールを使用すると、**pcp** の設定とデプロイメントが Playbook によって処理されるため、**pcp** を個別に設定せずに、**pcp** を使用してシステムパフォーマンスを監視できます。

表22.1 metrics 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**

注記

metrics_connections で使用されるパラメーターの詳細と、**metrics** システムロールに関する追加情報は、**/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** ファイルを参照してください。

22.2. METRICS システムロールを使用した視覚化によるローカルシステムの監視

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、**Grafana** でデータ可視化を同時にプロビジョニングする方法を説明します。

前提条件

- Ansible Core パッケージがコントロールマシンにインストールされている。
- 監視するマシンに **rhel-system-roles** パッケージがインストールされている。

手順

1. 以下のコンテンツをインベントリーに追加して、**/etc/ansible/hosts** Ansible インベントリーの **localhost** を設定します。

```
localhost ansible_connection=local
```

2. 以下の内容を含む Ansible Playbook を作成します。

```
---
- name: Manage metrics
  hosts: localhost
  vars:
    metrics_graph_service: yes
    metrics_manage_firewall: true
    metrics_manage_selinux: true
  roles:
    - rhel-system-roles.metrics
```

3. Ansible Playbook の実行:

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

注記

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ファイアウォールと 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。

4. 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 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。

22.3. METRICS システムロールを使用した自己監視のための個別システムフリートの設定

この手順では、**metrics** システムロールを使用して、それ自体を監視するマシンフリートの設定方法を説明します。

前提条件

- Ansible Core パッケージがコントロールマシンにインストールされている。
- Playbook の実行に使用するマシンに **rhel-system-roles** パッケージがインストールされている。
- SSH 接続が確立している。

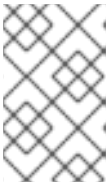
手順

1. Playbook 経由で監視するマシンの名前または IP を、括弧で囲まれた識別グループ名の下にある **/etc/ansible/hosts** Ansible インベントリーファイルに追加します。

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

2. 以下の内容を含む Ansible Playbook を作成します。

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



注記

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** 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。

3. Ansible Playbook の実行:

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

リモートシステムに接続するためのパスワードを求められる **-k** です。

22.4. METRICS システムロールを使用したローカルマシン経由でのマシンフリートの一元監視

この手順では、**grafana** を介したデータの視覚化のプロビジョニングおよび **redis** 経由でのデータのクエリーをしながら、**metrics** システムロールを使用して、マシンフリートを一元管理するローカルマシンの設定方法を説明します。

前提条件

- Ansible Core パッケージがコントロールマシンにインストールされている。
- Playbook の実行に使用するマシンに **rhel-system-roles** パッケージがインストールされている。

手順

1. 以下の内容を含む Ansible Playbook を作成します。

```
---
- hosts: localhost
  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
  roles:
    - rhel-system-roles.metrics
```

2. Ansible Playbook の実行:

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

注記

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。

3. マシンによって一元的に収集されるメトリック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。

22.5. METRICS システムロールを使用したシステム監視中の認証設定

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を説明します。

前提条件

- Ansible Core パッケージがコントロールマシンにインストールされている。
- Playbook の実行に使用するマシンに **rhel-system-roles** パッケージがインストールされている。

手順

1. 認証を設定する Ansible Playbook に、以下の変数を追加します。

```
---
  vars:
    metrics_username: your_username
    metrics_password: your_password
    metrics_manage_firewall: true
    metrics_manage_selinux: true
```

注記

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. Ansible Playbook の実行:

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

検証手順

- **sasl** 設定を確認します。

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

ip_adress は、ホストの IP アドレスに置き換える必要があります。

22.6. METRICS システムロールを使用した SQL サーバーのメトリクスコレクションの設定と有効化

この手順では、**metrics** RHEL システムロールを使用して、ローカルシステムの **pcp** を使用して Microsoft SQL Server のメトリック収集の設定と有効化を自動化する方法を説明します。

前提条件

- Ansible Core パッケージがコントロールマシンにインストールされている。
- 監視するマシンに **rhel-system-roles** パッケージがインストールされている。
- Red Hat Enterprise Linux に Microsoft SQL Server をインストールし、SQL Server への信頼できる接続を確立している。Red Hat に SQL Server をインストールしてデータベースを作成する を参照してください。
- 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。Red Hat Enterprise Server および Oracle Linux を参照してください。

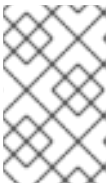
手順

1. 以下のコンテンツをインベントリーに追加して、**/etc/ansible/hosts** Ansible インベントリーの **localhost** を設定します。

```
localhost ansible_connection=local
```

2. 以下の内容が含まれる Ansible Playbook を作成します。

```
---
- hosts: localhost
  vars:
   metrics_from_mssql: true
   metrics_manage_firewall: true
   metrics_manage_selinux: true
  roles:
   - role: rhel-system-roles.metrics
```



注記

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。

3. Ansible Playbook の実行:

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

■

検証手順

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

関連情報

- Microsoft SQL Server での Performance Co-Pilot の使用に関する詳細は、Red Hat Developers Blog を参照してください。

第23章 MICROSOFT.SQL.SERVER ANSIBLE ロールを使用した MICROSOFT SQL SERVER の設定

管理者は、**microsoft.sql.server** Ansible ロールを使用して、Microsoft SQL Server (SQL Server) をインストール、設定、および起動できます。**microsoft.sql.server** Ansible ロールは、オペレーティングシステムを最適化して、SQL Server のパフォーマンスとスループットを向上させます。このロールは、SQL Server ワークロードを実行するために推奨される設定を使用し、RHEL ホストの設定を簡素化して自動化します。

23.1. 前提条件

- 2GB の RAM
- SQL Server を設定するマネージドノードへの **root** アクセス
- 事前設定済みファイアウォール
  ロールがファイアウォールを自動的に管理できるように、**mssql_manage_firewall** 変数を **true** に設定できます。

  または、**mssql_tcp_port** 変数で設定された SQL Server TCP ポートで接続を有効にします。この変数を定義しないと、ロールのデフォルトは TCP ポート番号 **1433** になります。

  新しいポートを追加するには、次のコマンドを実行します。

  ```
  # firewall-cmd --add-port=xxxx/tcp --permanent
  # firewall-cmd --reload
  ```

  xxx を TCP ポート番号に置き換え、ファイアウォールルールを再読み込みします。
- **オプション** - SQL ステートメントと、それを SQL Server に入力するプロシージャーを含む、**.sql** 拡張子を持つファイルを作成します。

23.2. MICROSOFT.SQL.SERVER ANSIBLE ロールのインストール

microsoft.sql.server Ansible ロールは、 **ansible-collection-microsoft-sql** パッケージに含まれています。

前提条件

- **root** アクセス

手順

1. RHEL 7.9 AppStream リポジトリーで利用可能な Ansible Core をインストールします。

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

2. **microsoft.sql.server** Ansible ロールをインストールします。

   ```
   # *yum install ansible-collection-microsoft-sql*
   ```

23.3. MICROSOFT.SQL.SERVER ANSIBLE ロールを使用した SQL サーバーのインストールと設定

microsoft.sql.server Ansible ロールを使用して、SQL サーバーをインストールおよび設定できます。

前提条件

- Ansible インベントリーが作成されます。

手順

1. 拡張子が **.yml** のファイルを作成します。例: **mssql-server.yml**
2. 以下の内容を **.yml** に追加します。

```
---
- hosts: all
  vars:
    mssql_accept_microsoft_odbc_driver_17_for_sql_server_eula: true
    mssql_accept_microsoft_cli_utilities_for_sql_server_eula: true
    mssql_accept_microsoft_sql_server_standard_eula: true
    mssql_password: <password>
    mssql_edition: Developer
    mssql_tcp_port: 1433
  roles:
    - microsoft.sql.server
```

   <password> を、使用している SQL Server のパスワードに置き換えます。

3. **mssql-server.yml** Ansible Playbook を実行します。

```
# *ansible-playbook mssql-server.yml*
```

23.4. TLS 変数

次の変数を使用して、トランスポートレベルセキュリティー (TLS) プロトコルを設定できます。

表23.1 TLS ロール変数

ロール変数	説明

ロール変数	説明
mssql_tls_enable	この変数は、TLS 暗号化を有効または無効にします。 **microsoft.sql.server** Ansible ロールは、変数が **true** に設定されていると、以下のタスクを実行します。 ● SQL Server の **/etc/pki/tls/certs/** に TLS 証明書をコピーします。 ● 秘密鍵を SQL Server 上の **/etc/pki/tls/private/** にコピーします。 ● TLS 証明書および秘密鍵を使用して接続を暗号化するように SQL Server を設定します。 **false** に設定すると、TLS 暗号化が無効になります。このロールは、既存の証明書および秘密鍵ファイルを削除しません。
mssql_tls_cert	この変数を定義するには、TLS 証明書ファイルのパスを入力します。
mssql_tls_private_key	この変数を定義するには、秘密鍵ファイルのパスを入力します。
mssql_tls_remote_src	ロールが **mssql_tls_cert** ファイルと **mssql_tls_private_key** ファイルをリモートで検索するか、コントロールノードで検索するかを定義します。 デフォルトの **false** に設定すると、ロールは Ansible コントロールノードで **mssql_tls_cert** または **mssql_tls_private_key** ファイルを検索します。 **true** に設定すると、ロールは Ansible 管理対象ノードで **mssql_tls_cert** または **mssql_tls_private_key** ファイルを検索します。
mssql_tls_version	この変数を定義して、使用する TSL バージョンを選択します。 デフォルト値は **1.2** です。

ロール変数	説明
mssql_tls_force	この変数を **true** に設定すると、ホストコンピューターの証明書および秘密鍵のファイルが置き換えられます。ファイルは、**/etc/pki/tls/certs/** ディレクトリーおよび **/etc/pki/tls/private/** ディレクトリーの下に存在している必要があります。 デフォルトは **false** です。

23.5. MLSERVICES の EULA への同意

必要な SQL Server Machine Learning Services (MLServices) をインストールするには、Python パッケージおよび R パッケージのオープンソースディストリビューション用のすべての EULA に同意する必要があります。

使用許諾条件は、**/usr/share/doc/mssql-server** を参照してください。

表23.2 SQL Server Machine Learning Services の EULA 変数

ロール変数	説明
mssql_accept_microsoft_sql_server_standard_eula	この変数は、**mssql-conf** パッケージのインストールに関する条件に同意するかどうかを決定します。 条件に同意する場合は、この変数を **true** に設定します。 デフォルトは **false** です。

23.6. MICROSOFT ODBC 17 向け EULA への同意

すべての EULA に同意し、Microsoft Open Database Connectivity (ODBC) ドライバーをインストールする必要があります。

使用許諾条件については、**/usr/share/doc/msodbcsql17/LICENSE.txt** および **/usr/share/doc/mssql-tools/LICENSE.txt** を参照してください。

表23.3 Microsoft ODBC 17 EULA 変数

ロール変数	説明
mssql_accept_microsoft_odbc_driver_17_for_sql_server_eula	この変数は、**msodbcsql17** パッケージのインストールに関する条件に同意するかどうかを決定します。 条件に同意する場合は、この変数を **true** に設定します。 デフォルトは **false** です。

ロール変数	説明
mssql_accept_microsoft_cli_utilities_for_sql_server_eula	この変数は、**mssql-tools** パッケージのインストールに関する条件に同意するかどうかを決定します。 条件に同意する場合は、この変数を **true** に設定します。 デフォルトは **false** です。

23.7. 高可用性変数

次の表の変数を使用して、Microsoft SQL Server の高可用性を設定できます。

表23.4 高可用性設定変数

変数	説明
mssql_ha_configure	デフォルト値は **false** です。 **true** に設定すると、次のアクションが実行されます。 ● **mssql_ha_listener_port** 変数からポートを開いてファイアウォールを設定し、ファイアウォールで **high-availability** サービスを有効にします。 ● 高可用性のために SQL Server を設定します。 ○ Always On Health イベントを有効にします。 ○ プライマリーレプリカで証明書を作成し、それを他のレプリカに配布します。 ○ エンドポイントと可用性グループを設定します。 ○ Pacemaker の **mssql_ha_login** 変数からユーザーを設定します。 ● オプション: Pacemaker を設定するシステムロールの **ha_cluster** ロールが含まれます。**mssql_ha_cluster_run_role** を **true** に設定し、**ha_cluster** ロールが Pacemaker クラスター設定に必要とするすべての変数を提供する必要があります。

変数	説明
mssql_ha_replica_type	この変数は、ホストで設定できるレプリカのタイプを指定します。この変数は、**primary**、**synchronous**、および **witness** に設定できます。1 つのホストでのみ **primary** に設定する必要があります。
mssql_ha_listener_port	デフォルトのポートは **5022** です。 ロールは、この TCP ポートを使用して、Always On 可用性グループのデータをレプリケートします。
mssql_ha_cert_name	Always On 可用性グループのメンバー間のトランザクションを保護するには、証明書の名前を定義する必要があります。
mssql_ha_master_key_password	証明書で使用するマスターキーのパスワードを設定する必要があります。
mssql_ha_private_key_password	証明書で使用する秘密鍵のパスワードを設定する必要があります。
mssql_ha_reset_cert	デフォルト値は **false** です。 **true** に設定されている場合、Always On 可用性グループが使用する証明書をリセットします。
mssql_ha_endpoint_name	設定するエンドポイントの名前を定義する必要があります。
mssql_ha_ag_name	設定する可用性グループの名前を定義する必要があります。
mssql_ha_db_names	レプリケートするデータベースのリストを定義できます。それ以外の場合、ロールはデータベースをレプリケートせずにクラスターを作成します。
mssql_ha_login	SQL Server Pacemaker リソースエージェントは、このユーザーを使用してデータベースの正常性チェックを実行し、レプリカからプライマリーサーバーへの状態遷移を管理します。
mssql_ha_login_password	SQL Server の **mssql_ha_login** ユーザーのパスワード。

変数	説明
mssql_ha_cluster_run_role	デフォルト値は **false** です。 この変数は、このロールが **ha_cluster** ロールを実行するかどうかを定義します。 **ha_cluster** ロールは、指定されたノード上の HA クラスターの設定を置き換えることに注意してください。HA クラスター用に現在設定されている変数はすべて消去され、上書きされます。 この制限を回避するために、**microsoft.sql.server** ロールは、既存の Pacemaker 設定を上書きしないように、**ha_cluster** ロールの変数を設定しません。 **microsoft.sql.server** で **ha_cluster** ロールを実行する場合は、この変数を **true** に設定し、**microsoft.sql.server** ロール呼び出しで **ha_cluster** ロールの変数を指定します。

このロールは、データベースを **/var/opt/mssql/data/** ディレクトリーにバックアップすることに注意してください。

Microsoft SQL Server で高可用性変数を使用する方法の例:

- Automation Hub からロールをインストールする場合は、サーバー上の **~/.ansible/collections/ansible_collections/microsoft/sql/roles/server/README.md** ファイルを参照してください。
- パッケージからロールをインストールする場合は、ブラウザーで **/usr/share/microsoft/sql-server/README.html** ファイルを開きます。

第24章 TLOG RHEL システムロールを使用したセッションの記録用のシステムの設定

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 で端末セッションを録画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 での端末セッションの録画用に RHEL システムを設定できます。

SSSD サービスを使用して、ユーザーごと、またはユーザーグループごとに録画を行うように設定できます。

関連情報

- RHEL でのセッションの録画に関する詳細は、Recording Sessions を参照してください。

24.2. TLOG システムロールのコンポーネントおよびパラメーター

セッションの録画ソリューションには、以下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以下のとおりです。

ロール変数	説明
tlog_use_sssd (default: yes)	SSSD を使用してセッションの録画を設定します (録画したユーザーまたはグループの管理方法として推奨)。
tlog_scope_sssd (default: none)	SSSD 録画スコープの設定: all / some / none
tlog_users_sssd (default: [])	録画するユーザーの YAML リスト
tlog_groups_sssd (default: [])	録画するグループの YAML リスト

- **tlog** で使用されるパラメーターの詳細と、**tlog** システムロールに関する追加情報は、**/usr/share/ansible/roles/rhel-system-roles.tlog/README.md** ファイルを参照してください。

24.3. TLOG RHEL システムロールのデプロイ

以下の手順に従って、Ansible Playbook を準備および適用し、RHEL システムが systemd ジャーナルにセッションの録画データをログに記録するように設定します。

前提条件

- コントロールノードから **tlog** システムロールが設定されるターゲットシステムへアクセスするための SSH キーを設定している。
- **tlog** システムロールを設定するシステムが 1 つ以上ある。
- Ansible Core パッケージがコントロールマシンにインストールされている。
- **rhel-system-roles** パッケージがコントロールマシンにインストールされている。

手順

1. 以下の内容を含む新しい **playbook.yml** ファイルを作成します。

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

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

詳細は以下のようになります。

- **tlog_scope_sssd**:
  - **some** は、**all** または **none** ではなく、特定のユーザーおよびグループのみを録画することを指定します。
- **tlog_users_sssd**:
  - **recorded-user** は、セッションを録画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. オプション: Playbook の構文を確認します。

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

3. インベントリーファイルで Playbook を実行します。

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

その結果、Playbook は指定したシステムに **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、Web コンソールインターフェイスで録画を表示および再生できるようにする **Cockpit** モジュールです。

検証手順

システムで SSSD 設定ドロップファイルが作成されることを確認するには、以下の手順を実行します。

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。

24.4. グループまたはユーザーの一覧を除外するための TLOG RHEL システムロールのデプロイ

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。以下の手順に従って、Ansible Playbook を準備および適用し、ユーザーまたはグループがセッションを録画して systemd ジャーナルにロギングしないように RHEL システムを設定します。

前提条件

- コントロールノードから **tlog** システムロールを設定するターゲットシステムへアクセスするための SSH キーを設定している。
- **tlog** システムロールを設定するシステムが 1 つ以上ある。
- Ansible Core パッケージがコントロールマシンにインストールされている。
- **rhel-system-roles** パッケージがコントロールマシンにインストールされている。

手順

1. 以下の内容を含む新しい **playbook.yml** ファイルを作成します。

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

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

詳細は以下のようになります。

- **tlog_scope_sssd**:
  - **all**: ユーザーおよびグループをすべて録画するように指定します。

- **tlog_exclude_users_sssd**:
  - User name: セッションの録画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**:
  - **admins** は、セッションの録画から除外するグループを指定します。

2. オプションで Playbook の構文を確認します。

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

3. インベントリーファイルで Playbook を実行します。

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

その結果、Playbook は指定したシステムに **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、Web コンソールインターフェイスで録画を表示および再生できるようにする **Cockpit** モジュールです。

検証手順

システムで SSSD 設定ドロップファイルが作成されることを確認するには、以下の手順を実行します。

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。

関連情報

- **/usr/share/doc/rhel-system-roles/tlog/** ディレクトリーおよび **/usr/share/ansible/roles/rhel-system-roles.tlog/** ディレクトリーを参照してください。
- Recording a session using the deployed Terminal Session Recording System Role in the CLI 。

24.5. CLI でデプロイされた TLOG システムロールを使用したセッションの録画

指定したシステムに **tlog** システムロールをデプロイしたら、コマンドラインインターフェイス (CLI) を使用してユーザー端末セッションを録画できます。

前提条件

- ターゲットシステムに **tlog** システムロールをデプロイしている。
- SSSD 設定ドロップファイルが **/etc/sssd/conf.d** ディレクトリーに作成されていること。Deploying the Terminal Session Recording RHEL System Role を参照してください。

手順

1. ユーザーを作成し、このユーザーにパスワードを割り当てます。

```
# useradd recorded-user
# passwd recorded-user
```

2. 作成したユーザーとしてシステムにログインします。

```
# ssh recorded-user@localhost
```

3. 認証用に yes または no を入力するようにシステムが求めたら、yes を入力します。
4. **recorded-user** の パスワードを挿入します。
システムは、録画しているセッションに関するメッセージを表示します。

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

5. セッションの録画が完了したら、以下を入力します。

```
# exit
```

システムはユーザーからログアウトし、ローカルホストとの接続を閉じます。

これにより、ユーザーセッションは録画および保存され、ジャーナルを使用して再生することができます。

検証手順

ジャーナルで録画したセッションを表示するには、以下の手順を実施します。

1. 以下のコマンドを実行します。

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

2. 録画したジャーナルエントリー **tlog-rec** の **MESSAGE** フィールドを検索します。

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

24.6. CLI を使用した録画したセッションの表示

コマンドラインインターフェイス (CLI) を使用して、ジャーナルからユーザーセッションの録画を再生できます。

前提条件

- ユーザーセッションを録画している。CLI でデプロイされた tlog システムロールを使用したセッションの録画 を参照してください。

手順

1. CLI 端末で、ユーザーセッションの録画を再生します。

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

2. **tlog** 録画を検索します。

```
$ /tlog-rec
```

以下のような詳細が表示されます。

- ユーザーセッションの録画用のユーザー名
- **out_txt** フィールド (録画したセッションの raw 出力エンコード)
- 識別子番号 TLOG_REC=**ID_number**

3. 識別子番号 TLOG_REC=**ID_number** をコピーします。
4. 識別子番号 TLOG_REC=**ID_number** を使用して録画を再生します。

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

これにより、ユーザーセッションの録画端末の出力が再生されることがわかります。

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以下のとおりです。

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である **true** に設定されている場合は、クラスターメンバーとして使用するシステムで 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必要です。そうでない場合、システムロールは失敗します。

ha_cluster_manage_firewall

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 7.9 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

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。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、パスワードの暗号化を行うことが推奨されます。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_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

certificate システムロールを使用して、**pcsd** 秘密キーと証明書を作成します。
システムが **pcsd** 秘密キーと証明書を使用して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、定義に従って **pcsd** の秘密キー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い場合は、**ha_cluster** システムロールは **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ている変数 **certificate_requests** の値に設定されます。**certificate** システムロールの詳細は、RHEL System Roles 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、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

ルは自己署名証明書を作成します。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

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) マニュアルページの **cluster** セクションの **セットアップ** 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** System Role 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

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) マニュアルページの **cluster** セクションの **セットアップ** 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** System Role 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

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) -

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

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値フラグ。この変数のデフォルト値は **false** です。
SBD を有効にする **ha_cluster** System Role 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

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 を有効にする **ha_cluster** System Role Playbook の例については、Configuring a high availability cluster with SBD node fencing 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については、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。
- **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以下のとおりです。

```
- 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
  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以下のとおりです。

```
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_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
```

リソースに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 8.8 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。

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
```

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

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。すべての SBD デバイスを共有し、すべてのノードから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 変数については、**ha_cluster** システムロール変数 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** - 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: /dev/watchdog2
        sbd_devices:
          - /dev/vdx
          - /dev/vdy
    node2:
      ha_cluster:
        sbd_watchdog: /dev/watchdog1
```

```
sbd_devices:
  - /dev/vdw
  - /dev/vdz
```

25.3. 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

ha_cluster システムロールを使用して、高可用性クラスター内に 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** システムロールは **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。

前提条件

- **ansible-core** パッケージと **rhel-system-roles** パッケージが、Playbook を実行するノードにインストールされている。

注記

ansible-core をクラスターメンバーノードにインストールする必要はありません。

- 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。

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。

手順

1. 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、クラスター内のノードを指定するインベントリーファイルを作成します。
2. Playbook ファイルを作成します (例: **new-cluster.yml**)。

注記

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。

次の Playbook ファイルの例では、**firewalld** サービスと **selinux** サービスを実行するクラスターを設定し、自己署名 **pcsd** 証明書と秘密キーファイルを **/var/lib/pcsd** に作成します。**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。

```
- hosts: node1 node2
  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
  roles:
    - linux-system-roles.ha_cluster
```

3. ファイルを保存します。
4. 手順 1 で作成したインベントリーファイル **inventory** へのパスを指定して、Playbook を実行します。

```
# ansible-playbook -i inventory new-cluster.yml
```

関連情報

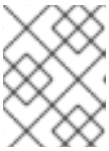
- RHEL システムロールを使用した証明書の要求

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。

前提条件

- Playbook を実行するノードに **ansible-core** がインストールされている。



注記

ansible-core をクラスターメンバーノードにインストールする必要はありません。

- Playbook を実行するシステムに **rhel-system-roles** パッケージがインストールされている。
- 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。

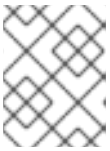
警告

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。

手順

1. 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、クラスター内のノードを指定するインベントリーファイルを作成します。

2. Playbook ファイルを作成します (例: **new-cluster.yml**)。



注記

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。

次の Playbook ファイルの例では、フェンシングが設定されておらず、リソースも実行されていない、**firewalld** サービスと **selinux** サービスを実行するクラスターを設定します。

```
- hosts: node1 node2
  vars:
    ha_cluster_cluster_name: my-new-cluster
    ha_cluster_hacluster_password: password
    ha_cluster_manage_firewall: true
    ha_cluster_manage_selinux: true

  roles:
    - rhel-system-roles.ha_cluster
```

3. ファイルを保存します。

4. 手順 1 で作成したインベントリーファイル **inventory** へのパスを指定して、Playbook を実行します。

```
# ansible-playbook -i inventory new-cluster.yml
```

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。

前提条件

- Playbook を実行するノードに **ansible-core** がインストールされている。

注記

ansible-core をクラスターメンバーノードにインストールする必要はありません。

- Playbook を実行するシステムに **rhel-system-roles** パッケージがインストールされている。
- 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。

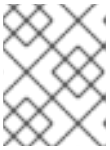
警告

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。

手順

1. 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、クラスター内のノードを指定するインベントリーファイルを作成します。

2. Playbook ファイルを作成します (例: **new-cluster.yml**)。



注記

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。

次の Playbook ファイルの例では、**firewalld** サービスと **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。

```
- hosts: node1 node2
  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

roles:
  - rhel-system-roles.ha_cluster
```

3. ファイルを保存します。

4. 手順1で作成したインベントリーファイル **inventory** へのパスを指定して、Playbook を実行します。

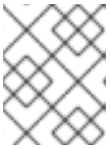
```
# ansible-playbook -i inventory new-cluster.yml
```

25.6. 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。

前提条件

- Playbook を実行するノードに **ansible-core** がインストールされている。



注記

ansible-core をクラスターメンバーノードにインストールする必要はありません。

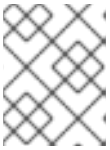
- Playbook を実行するシステムに **rhel-system-roles** パッケージがインストールされている。
- クラスターメンバーとして使用するシステムには、RHEL および RHEL High Availability Add-On のアクティブなサブスクリプションがある。

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。

手順

1. 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、クラスター内のノードを指定するインベントリーファイルを作成します。
2. Playbook ファイルを作成します (例: **new-cluster.yml**)。



注記

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。

次の Playbook ファイルの例では、**firewalld** サービスと **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。

```
- hosts: node1 node2
  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

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

3. ファイルを保存します。
4. 手順 1 で作成したインベントリーファイル **inventory** へのパスを指定して、Playbook を実行します。

```
# ansible-playbook -i inventory new-cluster.yml
```

25.7. 高可用性クラスターでの COROSYNC 値の設定

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。

前提条件

- Playbook を実行するノードに **ansible-core** がインストールされている。

注記

ansible-core をクラスターメンバーノードにインストールする必要はありません。

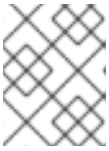
- Playbook を実行するシステムに **rhel-system-roles** パッケージがインストールされている。
- クラスターメンバーとして使用するシステムには、RHEL および RHEL High Availability Add-On のアクティブなサブスクリプションがある。

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。

手順

1. 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、クラスター内のノードを指定するインベントリーファイルを作成します。
2. Playbook ファイルを作成します (例: **new-cluster.yml**)。



注記

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。

次の Playbook ファイルの例では、Corosync プロパティーを設定する **firewalld** サービスと **selinux** サービスを実行するクラスターを設定します。

```
- hosts: node1 node2
  vars:
    ha_cluster_cluster_name: my-new-cluster
    ha_cluster_hacluster_password: password
    ha_cluster_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
```

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

3. ファイルを保存します。
4. 手順 1 で作成したインベントリーファイル **inventory** へのパスを指定して、Playbook を実行します。

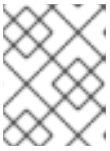
```
# ansible-playbook -i inventory new-cluster.yml
```

25.8. 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

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。

前提条件

- Playbook を実行するノードに **ansible-core** がインストールされている。



注記

ansible-core をクラスターメンバーノードにインストールする必要はありません。

- Playbook を実行するシステムに **rhel-system-roles** パッケージがインストールされている。
- 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。

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。

手順

1. 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、クラスター内のノードを指定するインベントリーファイルを作成します。必要に応じて、クラスター内の各ノードのウォッチドッグデバイスと SBD デバイスをインベントリーファイルで設定できます。
2. Playbook ファイルを作成します (例: **new-cluster.yml**)。

注記

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。

次の Playbook ファイルの例では、SBD フェンシングを使用する **firewalld** サービスと **selinux** サービスを実行するクラスターを設定します。

```
- hosts: node1 node2
  vars:
    ha_cluster_cluster_name: my-new-cluster
    ha_cluster_hacluster_password: password
    ha_cluster_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: 5

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

3. ファイルを保存します。

4. 手順 1 で作成したインベントリーファイル **inventory** へのパスを指定して、Playbook を実行します。

```
# ansible-playbook -i inventory new-cluster.yml
```

25.9. 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

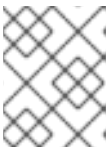
ha_cluster システムロールを使用し、別個のクォーラムデバイスを使用した高可用性クラスターを設定するには、まずクォーラムデバイスをセットアップします。クォーラムデバイスをセットアップした後は、任意の数のクラスターでデバイスを使用できます。

25.9.1. 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。

前提条件

- **ansible-core** パッケージと **rhel-system-roles** パッケージが、Playbook を実行するノードにインストールされている。



注記

ansible-core をクラスターメンバーノードにインストールする必要はありません。

- 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。

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。

手順

1. Playbook ファイル (例: **qdev-playbook.yml**) を作成します。

注記

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。

次の Playbook ファイルの例では、**firewalld** サービスと **selinux** サービスを実行しているシステム上でクォーラムデバイスを設定します。

```
- hosts: nodeQ
  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

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

2. ファイルを保存します。

3. クォーラムデバイスのホストノードを指定して、Playbook を実行します。

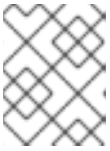
```
# ansible-playbook -i nodeQ, qdev-playbook.yml
```

25.9.2. 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。

前提条件

- Playbook を実行するノードに **ansible-core** がインストールされている。



注記

ansible-core をクラスターメンバーノードにインストールする必要はありません。

- Playbook を実行するシステムに **rhel-system-roles** パッケージがインストールされている。
- 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。
- クォーラムデバイスが設定されている。

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。

手順

1. 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、クラスター内のノードを指定するインベントリーファイルを作成します。
2. Playbook ファイルを作成します (例: **new-cluster.yml**)。

注記

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。

次の Playbook ファイルの例では、クォーラムデバイスを使用する **firewalld** サービスと **selinux** サービスを実行するクラスターを設定します。

```
- hosts: node1 node2
  vars:
    ha_cluster_cluster_name: my-new-cluster
    ha_cluster_hacluster_password: password
    ha_cluster_manage_firewall: true
    ha_cluster_manage_selinux: true
    ha_cluster_quorum:
      device:
        model: net
        model_options:
          - name: host
            value: nodeQ
          - name: algorithm
            value: lms

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

3. ファイルを保存します。
4. 手順 1 で作成したインベントリーファイル **inventory** へのパスを指定して、Playbook を実行します。

```
# ansible-playbook -i inventory new-cluster.yml
```

25.10. 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。

前提条件

- Playbook を実行するノードに **ansible-core** がインストールされている。

注記

ansible-core をクラスターメンバーノードにインストールする必要はありません。

- Playbook を実行するシステムに **rhel-system-roles** パッケージがインストールされている。
- 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。
- システムに Apache に必要なパブリック仮想 IP アドレスが含まれている。
- システムには、iSCSI、ファイバーチャネル、またはその他の共有ネットワークブロックデバイスを使用する、クラスターのノードに対する共有ストレージが含まれている。
- 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。

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。

手順

1. 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、クラスター内のノードを指定するインベントリーファイルを作成します。
2. Playbook ファイルを作成します (例: **http-cluster.yml**)。

注記

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。

次の Playbook ファイルの例では、**firewalld** サービスと **selinux** サービスを実行するアクティブ/パッシブの 2 ノード HA クラスター内に、以前作成した 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。

```
- 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
```

3. ファイルを保存します。

4. 手順 1 で作成したインベントリーファイル **inventory** へのパスを指定して、Playbook を実行します。

```
# ansible-playbook -i inventory http-cluster.yml
```

5. **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
```

検証手順

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
```

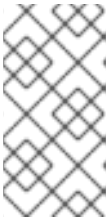
4. ノード **z1** を **standby** モードにしたら、クラスター内のノードのいずれかからクラスターのステータスを確認します。リソースはすべて **z2** で実行しているはずです。

```
[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。

25.11. 関連情報

- 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
- **rhel-system-roles** パッケージでインストールされたドキュメント (**/usr/share/ansible/roles/rhel-system-roles.logging/README.html**)
- RHEL システムロール のナレッジベース記事
- **ansible-playbook(1)** の man ページ

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、秘密キーと証明書が生成されます。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.cockpit/README.md** ファイル
- Cockpit configuration file の 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** のシステムロールを呼び出す必要はありません。**cockpit** システムロールは、必要に応じてそれらを自動的に呼び出します。

前提条件

- 1 つ以上の **管理対象ノード** へのアクセスとパーミッション。
- **コントロールノード** へのアクセスおよびパーミッション。
  コントロールノードでは、
  - Red Hat Ansible Engine がインストールされている。
  - **rhel-system-roles** パッケージがインストールされている。
  - 管理ノードが記載されているインベントリーファイルが存在する。

手順

1. 以下の内容を含む新しい **playbook.yml** ファイルを作成します。

```
---
- hosts: all
  tasks:
    - name: Install RHEL web console
      include_role:
        name: rhel-system-roles.cockpit
      vars:
        cockpit_packages: default
        #cockpit_packages: minimal
        #cockpit_packages: full
        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. オプション: Playbook の構文を確認します。

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

3. インベントリーファイルで Playbook を実行します。

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

関連情報

- Installing and enabling the web console 。
- Requesting certificates using RHEL System Roles 。

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

podman RHEL システムロールを使用して、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。

関連情報

- RHEL システムロールのインストール
- **podman** で使用されるパラメーターの詳細と、**podman** RHEL システムロールに関する追加情報については、**/usr/share/ansible/roles/rhel-system-roles.podman/README.md** ファイルを参照してください。

27.2. 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以下のとおりです。

変数	説明

podman_kube_spec	管理する podman Pod と対応する 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開始します
	  ◦ **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。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。指定しない場合は、root が使用されます。

変数	説明
	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、root 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。

変数	説明
	○ 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-kubernetes.d/<application_name>.yml** ファイルにコピーされます。
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管理するには、ディレクトリーを絶対パス (root 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 root コンテナーの場合) として指定する必要があります。それ以外は無視されます。 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、ファイアウォール RHEL システムロールで使用される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** の **run_as_user** を使用してコンテナー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** の **run_as_group** を使用して、コンテナー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** の **systemd_unit_scope** を使用してコンテナー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、root 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 (**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、**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 (**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、**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 (**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、**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

- RHEL システムロールのインストール
- **podman** で使用されるパラメーターの詳細と、**podman** RHEL システムロールに関する追加情報については、**/usr/share/ansible/roles/rhel-system-roles.podman/README.md** ファイルを参照してください。

27.3. 関連情報

- **podman** で使用されるパラメーターの詳細と、**podman** RHEL システムロールに関する追加情報については、**/usr/share/ansible/roles/rhel-system-roles.podman/README.md** ファイルを参照してください。
- **ansible-playbook** コマンドの詳細は、man ページの **ansible-playbook(1)** を参照してください。

第28章 RHEL システムロールを使用した 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。

本章では、以下のトピックについて説明します。

- **ad_integration** システムロール
- ad_integration RHEL システムロールの変数
- **ad_integration** システムロールを使用した RHEL システムの AD への直接接続

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

- SSSD を使用して RHEL システムを AD に直接接続します 。

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。

ロール変数	説明
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**
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

28.3. AD_INTEGRATION システムロールを使用した RHEL システムの AD への直接接続

ad_integration システムロールを使用すると、Ansible Playbook を実行することで、RHEL システムと AD ドメイン間の直接統合を設定できます。

注記

RHEL8 以降、RHEL はデフォルトで RC4 暗号化をサポートしなくなりました。AD ドメインで AES を有効にできない場合は、**AD-SUPPORT** 暗号化ポリシーを有効にし、Playbook で RC4 暗号化を許可する必要があります。

重要

RHEL サーバーと AD 間の時刻は同期する必要があります。これを確実にするには、Playbook で **timesync** システムロールを使用します。

この例では、RHEL システムは、AD **Administrator** ユーザーと、Ansible コンテナーに保存されているこのユーザーのパスワードを使用して、**domain.example.com** AD ドメインに参加します。また、Playbook は **AD-SUPPORT** 暗号化ポリシーを設定し、RC4 暗号化を許可します。RHEL システムと AD 間の時刻同期を確実にするために、Playbook は **adserver.domain.example.com** サーバーを **timesync** ソースとして設定します。

前提条件

- 1 つ以上の **管理対象ノード** へのアクセスとパーミッション。
- **コントロールノード** へのアクセスおよびパーミッション。
  コントロールノードでは、
  - Red Hat Ansible Engine がインストールされている。
  - **rhel-system-roles** パッケージがインストールされている。
  - マネージドノードが記載されているインベントリーファイルがある。
- AD ドメインコントローラー上の次のポートが開いており、RHEL サーバーからアクセスできます。

表28.1 ad_integration システムロールを使用して Linux システムを AD に直接統合するために必要なポート

送信元ポート	送信先ポート	プロトコル	サービス
1024:65535	53	UDP および TCP	DNS
1024:65535	389	UDP および TCP	LDAP
1024:65535	636	TCP	LDAPS
1024:65535	88	UDP および TCP	Kerberos
1024:65535	464	UDP および TCP	Kerberos のパスワード変更/設定 (**kadmin**)
1024:65535	3268	TCP	LDAP グローバルカタログ
1024:65535	3269	TCP	LDAP グローバルカタログ SSL/TLS
1024:65535	123	UDP	NTP/Chrony (オプション)
1024:65535	323	UDP	NTP/Chrony (オプション)

手順

1. 次の内容を含む新しい **ad_integration.yml** ファイルを作成します。

```
---
- hosts: all
  vars:
    ad_integration_realm: "domain.example.com"
    ad_integration_password: !vault | vault encrypted password
```

```
    ad_integration_manage_crypto_policies: true
    ad_integration_allow_rc4_crypto: true
    ad_integration_timesync_source: "adserver.domain.example.com"
  roles:
    - linux-system-roles.ad_integration
---
```

2. オプション: Playbook の構文を確認します。

```
# ansible-playbook --syntax-check ad_integration.yml -i inventory_file
```

3. インベントリーファイルで Playbook を実行します。

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

検証

- **administrator** ユーザーなどの AD ユーザーの詳細を表示します。

```
getent passwd administrator@ad.example.com
administrator@ad.example.com:*:1450400500:1450400513:Administrator:/home/administrator
@ad.example.com:/bin/bash
```

28.4. 関連情報

- **/usr/share/ansible/roles/rhel-system-roles.ad_integration/README.md** ファイル
- **man ansible-playbook(1)**