Red Hat
Training and
Certification

Custom Course, Configure Identity Management with Ansible

Travis Michette

Petros Tselios <ptselios@redhat.com>

Table of Contents

Chapter 1. Manage IdM with Ansible

IdM Configuration with Ansible

Chapter 2. Objectives

After completing this section, you should be able to create and manage Ansible Playbooks and Ansible Inventories in Git repositories, following recommended practices.

2.1. Prerequisites

As soon as we have an IdM cluster installed, we can start it's configuration by using the relevant Ansible modules. In this section we will examine what are the requirements to do so, which options can be configured as well as an example playbook we will use to configure some basic parameters.

The prerequisites to use Ansible for the configuration of IdM are:

- You know the IdM administrator password.
- You have configured your Ansible control node to meet the following requirements:
  - You are using Ansible version 2.8 or later.
  - You have installed the ansible-freeipa package on the Ansible controller.
  - You have created an Ansible inventory file with the fully-qualified domain name (FQDN) of the IdM server.

Since we will need to use some secrets, like the administrator's password, we will use a vault to store those secrets. Thus, you must know the password of the vault file in order to use its variables in the playbooks.

2.2. How to structure your Ansible code and variables

There are various methodologies regarding the organization of Ansible playbooks, roles and inventories. In this section, we will discuss the benefits of using separate directories for Ansible roles, collections, and playbooks and the Ansible inventory.

By separating these resources, it can provide several advantages, such as easier management of sensitive information, simplified version control, and improved management of different operating environments. This approach can help reduce the risk of errors and simplify the management of complex IT infrastructures.

2.3. Create your inventory structure

Typically, an inventory file is named `inventory` or hosts and it's stored in the same directory as your playbooks. However, an inventory can nevertheless be also a directory containing:

- list(s) of hosts
- list(s) of groups, with sub-groups and hosts belonging to those groups
- dynamic inventory plug-ins configuration files
- dynamic inventory scripts (deprecated but still simple to use)

- structured host_vars directories
- structured group_vars directories

The recommendation is to start with such a structure and extend it step by step.

Having this in our mind, we can start working on our directory structure for both the inventory and the playbooks and roles that we will use for the IdM configuration.

Create the directories that will hold your inventory and the roles, playbooks, collections that you will use for the configuration of IdM:

```
[student@workstation ~]$  mkdir -p /home/student/workshop/idm-{inventory,code}
```

Create the structure of your inventory directory:

```
[student@workstation ~]$ cd  /home/student/workshop/idm-inventory
[student@workstation ~]$ mkdir ./{group,host}_vars
```

Following the recommended practices for the inventories, we can start with an inventory in a single file, in the `INI` format.

In case you want to use the same inventory file in order to deploy IdM Servers and to enroll clients using the `rhel_idm` collections, you must follow the naming conventions defined in the ansible-freeipa Github page.

Create an inventory file in INI format that will have at least one group and the IdM server:

```
[student@workstation ~]$ cat ./inventory
[all]
idm.lab.example.net
replica1.lab.example.net
replica2.lab.example.net

[rhidm_master]
idm.lab.example.net

[rhidm_replicas]
replica1.lab.example.net
replica2.lab.example.net

[rhidms:children]
rhidm_master
rhidm_replicas
```

2.3.1. Manage your variables

Ansible has 22 levels of variable precedence, which can be difficult to keep track of and may lead to unexpected behavior if not properly managed. Therefore, it is recommended to reduce the number of variable types to simplify and oversee the list of variables.

Moreover, the use of playbook variables is also not recommended as it blurs the separation between code and data. This applies to all constructs, including specific variable files as part of the play, such as "include_vars". Instead, it is best to use inventory variables, which represent your desired state. These variables have their own internal precedence, with group variables taking precedence over host variables.

In summary, simplifying your variables and using inventory variables instead of playbook variables can help avoid confusion and ensure a clear separation of code and data. Additionally, organizing your inventory directory can make it easier to manage and maintain.

Assuming you already have an inventory file, the next logical step is to create the appropriate directories under your inventory. This will help keep your inventory organized and easy to manage.

```
[student@workstation ~]$ cd  /home/student/workshop/idm-inventory
[student@workstation ~]$ mkdir -p ./group_vars/rhidm{s,_master,_replicas}
[student@workstation ~]$ mkdir -p ./host_vars/{idm,replica1,replica2}.lab.example.net
```

Organizing Ansible variables in separate files with appropriate names can help improve the organization and maintainability of your code. By separating variables into different files based on their purpose, you can easily locate and modify them when needed. This can also prevent errors caused by variables being defined in multiple places or with conflicting values. Additionally, using descriptive file names can make it easier for other team members to understand the purpose of the variables and their associated tasks. Overall, organizing variables in separate files with appropriate names can help make your Ansible code more organized, maintainable, and collaborative.

Naming conventions for Ansible variables are important for ensuring consistency and clarity in your code. One commonly used practice is to use **prefixes** to indicate the purpose or scope of the variable. For example, you might use "host_" to indicate a variable that applies to a specific host, or "groupname_" for a variable that applies to a specific group of hosts. This can help prevent conflicts between variables with similar names but different purposes.

Another good practice is to use **descriptive** names that reflect the intended purpose of the variable. For example, if a variable is used to specify the version of a software package, you might name it `package_version`. This can help make the code more understandable and make it easier for other team members to modify the code.

Additionally, it's important to avoid using reserved keywords or special characters in variable names, as this can cause errors or unexpected behavior. Overall, following consistent and descriptive naming conventions for Ansible variables can help make your code more readable, understandable, and maintainable.

2.3.2. Location of IdM related variables

When defining variables for an IdM configuration in Ansible, we have the option of defining them under the `rhidm_master` inventory group or under the `rhidms` group. By using the `rhidms` group, we gain more flexibility since we can use any of the IdM servers to configure our domain. This means that we are not limited to configuring our domain only on the master server, but can use any of the available servers in our inventory.

This flexibility is particularly useful in larger environments where we may have multiple IdM servers serving different functions. For example, we may have one server dedicated to user authentication and another server dedicated to managing DNS. By defining the variables under the `rhidms` group, we can easily configure the necessary components on each server without having to switch between different inventory groups.

It’s important to note that when defining variables under the `rhidms` group, we must ensure that the variables are applicable to all the servers in the group. This means that we must take into account any server-specific configuration that may be required. Additionally, we must ensure that we have defined the necessary variables for all the servers in the group before running the playbook, as any missing variables can cause the playbook to fail.

In summary, by defining variables under the `rhidms` inventory group, we gain more flexibility in configuring our IdM domain, allowing us to use any available server in our inventory. However, we must ensure that the variables are applicable to all servers in the group and that we have defined all the necessary variables before running the playbook.

2.4. Global IdM configuration options

As part of our global IdM configuration we want to:

- Define `/bin/bash` as the default shell for all users.
- Set the default group for new users
- Set the default e-mail domain
- Set the default domain resolution order

In RHEL 9 and later versions, it is possible to define parameters such as automatic SID creation for new users, the NETBIOS name of the server, or the addition of SID to existing users and groups.

Please refer to the Reference section for a list of all the global IdM options that can be modified by Ansible roles.

To modify the global IdM configuration options we use the `ansible-freeipa.config` module. The module uses the following variables for the items we want to modify:

- defaultshell
- defaultgroup
- emaildomain

- domain_resolution_order

Following the described good practices, we store the values of these variables in a file with a descriptive name, for example `rhidms_global_configuration.yml`

```
[student@workstation ~]$ cd  /home/student/workshop/idm-inventory/group_vars/rhidms
[student@workstation ~]$ cat ./rhidms_global_configuration.yml
---

# Default shell for new users
rhidm_defaultshell: "/bin/bash"

# Default group that all new users will be members to
rhidm_defaultgroup: "idmusers" ①

# Default email domain for the new users
rhidm_emaildomain: "example.net"

# Set list of domains used for short name qualification
rhidm_domain_resolution_order:
  - lab.example.net
  - example.net
...
```

① The default group for new users must already exists. This is defined here as an example.

2.5. Playbook or roles strategy

When it comes to configuring a system such as IdM with Ansible, there are various methodologies that one can follow. One such methodology is to create a playbook of playbooks, where a playbook calls other playbooks responsible for the configuration of various IdM topics. Each playbook in this scenario would call one or two roles to achieve a specific topic such as user and group management.

Another option is to create a playbook that calls a general-purpose IdM configuration playbook that in turn calls different roles. In this approach, we need to use Ansible tags to configure a specific topic. There are advantages and disadvantages to each option, and it is ultimately a matter of personal preference which one is used.

Both methodologies involve defining variables either under a specific inventory group or in a separate file. The advantage of creating a playbook of playbooks is that it allows for more fine-grained control over the configuration of specific topics, while the general-purpose playbook approach provides a more streamlined approach.

However, regardless of the chosen methodology, it is important to use Ansible tags to ensure that only the necessary tasks are executed. This can greatly speed up the configuration process and avoid unnecessary changes. Ultimately, the choice of methodology depends on the specific needs and preferences of the user, and both approaches can be effective for configuring IdM with Ansible.

In the following sections we will use the first approach. As one can read in the Automation structures, or how to name your playbooks, an IdM configuration playbook could be named as `+typerhidmconfigure.yml`

```
[student@workstation ~]$ cd  /home/student/workshop/idm-code
[student@workstation ~]$ cat ./type__rhidm__configure.yml
---
- name: Configure the RH IDM domain
  hosts: "{{ __config_host | groups['rhidms_master'][0] | default('localhost') }}" ①
  roles:
    - rhidm_global_configuration

...
```

① If the controller node is an IdM client, we can use it node for the configuration of IdM, otherwise we need to define another IdM client or server as the target node on which we will execute the playbook.

> Configuring global IdM settings using Ansible playbooks
>
> Ansible roles and modules for FreeIPA
>
> ansible-freeipa: Config Module
>
> Automation structures, or how to name your playbooks

Prepare your Ansible controller node

Chapter 3. Outcomes

You should be able to

- Prepare your workstation VM to act as a controller node for configuring IdM with Ansible
- Install packages needed by the Ansible modules
- Prepare the necessary directories structures and the inventory file
- Store secrets in vaults and create variable files for the global configuration

3.1. Prerequisites

If you did not reset classroom systems prior running this exercise, please reset them now. When your classroom systems are on-line again, logging on the `workstation` machine as the `student` user and run the following commands:

```
[student@workstation ~]$ lab ad-trust setup
[student@workstation ~]$ lab manage-clients setup
```

Chapter 4. Instructions

1) Install required packages

1.1) Install packages from lab's repositories

```
[student@workstation ~]$ sudo yum install -y git python-virtualenv gcc libffi-devel
openssl-devel

Transaction Summary

...output ommitted...

Is this ok [y/d/N]: y

Downloading Packages:

...output ommitted...

Complete!
```

1.2) Download `python2-babel` and `python2-jinja2` packages from the CentOS Community Build Service

```
[student@workstation ~]$ wget --no-check-certificate
https://cbs.centos.org/kojifiles/packages/babel/2.6.0/4.el7/noarch/python2-babel-
2.6.0-4.el7.noarch.rpm

[student@workstation ~]$ wget https://cbs.centos.org/kojifiles/packages/python-
jinja2/2.8.1/1.el7/noarch/python2-jinja2-2.8.1-1.el7.noarch.rpm --no-check-certificate
```

1.3) Install the python libraries

```
[student@workstation ~]$ sudo yum install -y \
  ./python2-jinja2-2.8.1-1.el7.noarch.rpm
  ./python2-babel-2.6.0-4.el7.noarch.rpm
[sudo] password for student: student

...output ommitted...

Complete!
```

2) Configure vim for Ansible

Configure `vim` to handle tabs as spaces and define tab space to 2 characters.

```
[student@workstation ~]$ vim ~/.vimrc
autocmd FileType yaml setlocal ai ts=2 sw=2 et
```

3) Create an Ansible Virtual Environment

Our classroom environment has an old version of Ansible which is not supported from the `ansible-freeipa` modules. Thus, we need to create a Python Virtual Environment on which we will install the necessary packages via `pip`.

Moreover, the version of RHEL is old and most of the python packages are outdated. Thus, we need to download, compile and install some of them.

```
[student@workstation ~]$ mkdir ~/venvs
[student@workstation ~]$ cd ~/venvs
[student@workstation venvs]$ *virtualenv ./ansible2.9 --system-site-packages *
Installing
Setuptools.....................................................................
..........................................................................done.
Installing
Pip............................................................................
..............................................done.
```

3) Use the virtual environment. **Download** a newer but compatible version of **pip**, **extract** the archive and **install** it manually.

```
[student@workstation venvs]$ source ./ansible2.9/bin/activate
(ansible2.9)[student@workstation venvs]$ wget -q
https://github.com/pypa/pip/archive/refs/tags/6.0.tar.gz -O pip-6.0.tar.gz

(ansible2.9)[student@workstation venvs]$ tar zxf pip-6.0.tar.gz

(ansible2.9)[student@workstation pip-6.0]$ cd ./pip-6.0
(ansible2.9)[student@workstation pip-6.0]$ python setup.py -q install
warning: no previously-included files found matching '.coveragerc'
...output ommitted...
no previously-included directories found matching 'tests'
```

4) Install and upgrade python libraries

```
(ansible2.9)[student@workstation venvs]$ cd ~/venvs
(ansible2.9)[student@workstation venvs]$ pip -q install setuptools==18.5
You are using pip version 6.0, however version 23.0.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command. ①
```

① You can safely ignore this message, we run a very old version of RHEL.

5) Install Ansible 2.9

```
(ansible2.9)[student@workstation venvs]$ pip -q install ansible==2.9
Collecting ansible==2.9
  Downloading ansible-2.9.0.tar.gz (14.1 MB)
...output ommitted...
Successfully installed ansible-2.9.0
```

5) Verify ansible's version

```
(ansible2.9)[student@workstation venvs]$ cd
(ansible2.9)[student@workstation ~]$ ansible --version
ansible 2.9.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/home/student/.ansible/plugins/modules',
u'/usr/share/ansible/plugins/modules']
  ansible python module location = /home/student/venvs/ansible2.9/lib/python2.7/site-
packages/ansible
  executable location = /home/student/venvs/ansible2.9/bin/ansible
  python version = 2.7.5 (default, May  3 2017, 07:55:04) [GCC 4.8.5 20150623 (Red Hat
4.8.5-14)]
```

6) Create the necessary directory structure for the automation

```
(ansible2.9)[student@workstation venvs]$ cd
(ansible2.9)[student@workstation ~]$ mkdir -p /home/student/workshop/idm-inventory
(ansible2.9)[student@workstation ~]$ mkdir -p /home/student/workshop/idm-code
```

7) Switch to the inventory directory and **create** its structure.

```
(ansible2.9)[student@workstation ~]$ cd /home/student/workshop/idm-inventory
(ansible2.9)[student@workstation idm-inventory]$ mkdir ./group_vars ./host_vars
```

8) **Create** the inventory file called `inventory` with the following information:

- A group named `rhidms` which will contain all RH IdM servers in our lab (idm, replica1, replica2)
- A group named `rhidm_master` which will contain the `idm` server
- A group named `rhidm_replicas` which will contain the `replica` and `replica2` servers
- A group named `all` which will contain all the IdM servers in our lab with their FQDN

```
(ansible2.9)[student@workstation idm-inventory]$ vim inventory
[all]
idm.lab.example.net
replica1.lab.example.net
```

```
replica2.lab.example.net

[rhidm_master]
idm.lab.example.net

[rhidm_replicas]
replica1.lab.example.net
replica2.lab.example.net

[rhidms:children]
rhidm_master
rhidm_replicas
```

9) **Test** the inventory and **connectivity** to the clients

```
(ansible2.9)[student@workstation idm-inventory]$ ansible -i ./inventory --graph
@all:
  |--@rhidms:
  |  |--@rhidm_master:
  |  |--|--idm.lab.example.net
  |  |--@rhidm_replicas:
  |  |--|--replica1.lab.example.net
  |  |--|--replica2.lab.example.net
  |--@ungrouped:

(ansible2.9)[student@workstation idm-inventory]$ ansible -i ./inventory -m ping rhidms

idm.lab.example.net| SUCCESS => {
    "changed": false,
    "ping": "pong"
}
replica1.lab.example.net| SUCCESS => {
    "changed": false,
    "ping": "pong"
}
replica2.lab.example.net| SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

10) **Create** a new directory under **group_vars** for the **rhidms** group.

```
(ansible2.9)[student@workstation idm-inventory]$ mkdir ./group_vars/rhidms
```

This concludes the section.

This concludes the section.

1. Clone the code repositories

```
(ansible2.9)[student@workstation idm-inventory]$ cd ~/workshop/idm-code
(ansible2.9)[student@workstation idm-code]$ git clone https://github.com/p-
tselios/idm-wshop.git ./
```

Prepare to use Ansible

Chapter 5. Objectives

After completing this section, you should be able to:

- Understand how to add Ansible collections to your local repository
- Understand how to use the Red Hat IDM Collection (ansible-freeipa)

Chapter 6. Ansible Configuration File (Overview)

6.1. Ansible Config

The **ansible.cfg** file controls how the **ansible** and **ansible-playbook** commands are run and interpreted. The configuration file has two (2) main sections that are commonly used, but include other sections as well. For the purpose of understanding how Ansible works, we will examine both the **[defaults]** section and the **[privilege_escalation]** section.

Listing 1. ansible.cfg Defaults Section

```
[defaults]
inventory = inventory ①
remote_user = devops ②
```

① Specifies which inventory file Ansible will use

② Specifies the remote user to be used by **ansible** or **ansible-playbook** commands.

A perfectly acceptable **ansible.cfg** might only have a [defaults] section specifying the inventory to be used.

The **[privilege_escalation]** of the Ansible configuration file defines how Ansible will run when additional privileges need to be provided. This often leverages `SUDO` and generally uses the **root** user, however, there are instances in which the privileged user is something other than root. The **become** and **become_ask_pass** options instruct Ansible whether it should attempt to become a privileged user by default as well as whether Ansible will prompt for the password.

Listing 2. ansible.cfg Privilege Escalation Section

```
[privilege_escalation]
become = False ①
become_method = sudo ②
become_user = root ③
become_ask_pass = False ④
```

① Sets default behavior whether to elevate privileges

② Sets method for privilege escalation

③ Sets username of privileged user

④ Sets option on whether or not user is prompted for password when perfoming privilege escalation.

Typically, the remote user we define in the ansible.cfg file can execute sudo commands without password. In our lab we lack this user, thus we will use `root` as the remote user.

Ansible Config File Precedence

- **ANSIBLE_CONFIG** - Environment Variable (highest)
- **ansible.cfg** - Config file in current working directory (most common and recommended)
- ~/**.ansible.cfg** - Ansible config file in the home directory
- **/etc/ansible/ansible.cfg** - Ansible's installed default location (lowest)

Chapter 7. Ansible Playbooks (Overview)

7.1. Ansible Playbooks

Ansible playbooks contain one or more tasks to execute against specified inventory nodes. Playbooks consist of one or more play and each play in a playbook consists of one or more tasks. Ansible playbooks and tasks are all about **key:value** pairs and **lists**. Understanding this basic format allows someone developing Ansible to form playbooks that are easier to create, troubleshoot/debug, and for someone else to understand.

7.2. Playbook Basics

An Ansible playbook is written/formatted in YAML so horizontal whitespace is critical and often the most troublesome part of debugging new Ansible playbooks. Playbooks have a general structure for the plays with directives such as: **name**, **hosts**, **vars**, **tasks**, and more. These play-level directives help form a readable structure much like **task-level** directives.

Listing 3. Play Structure Components

```
---
- name: install and start apache ①
  hosts: web ②
  become: yes ③

  tasks: ④
```

① Name of **play** in playbook

② List of hosts from inventory to execute play against **(required)**

③ Directive to override **ansible.cfg** and elevate privileges

④ Beginning of **tasks** section.

There can be other directives here, but at the most basic playbook, you will generally always see a **hosts** and a **tasks** section.

The first indentation level in a playbook denoted by **-** is the list of plays and this level will contain the **key:value** pairs that correspond to Ansible playbook directives. Understanding this and developing good habits and standards for indentations allows Ansible users to create playbook skeletons which help tremendously during the development/debugging cycle.

7.3. Running Playbooks

Playbooks can be run just like Ansible **ad-hoc** commands. In order to execute or run a playbook, it is necessary to use the **ansible-playbook** command and specify the playbook. The additional options available for the **ad-hoc** commands such as: **-e**, **-K**, **-b**, and others all still apply and perform the same functions when leveraged with the **ansible-playbook** command.

Chapter 8. Ansible Collections (Overview)

Ansible 2.9 introduced the concept of collections and provided mapping for Ansible modules that were moved into a collection namespace. Ansible 2.9 provided a mapping of the new module locations in collections and this mapping automatically works for existing Ansible playbooks in the initial versions of Ansible Automation Platform.

Ansible Module and Collection Mapping

https://github.com/ansible/ansible/blob/devel/lib/ansible/config/ansible_builtin_runtime.yml

8.1. Using Ansible Automation Platform Collections

Collections allowed development of Ansible core components to be separated from module and plug-in development. Upstream Ansible (the Ansible Project) unbundled modules from the Ansible core code beginning with Ansible Base (core) 2.10/2.11. Newer versions of Ansible require collections to be installed in order for modules to be available for Ansible. Playbooks should be developed using the **FQCNs** (fully-qualified connection names) when referring to modules in tasks. Existing playbooks can be fixed easily to work with collections, but it is better to re-write the playbooks to use the fully-qualified collection name (FCQN).

Downloading Collections

Collections modules can are brought into the Ansible project and leveraged via three ways:

1. A **requirements.yml** file in a sub-directory called **collections** within the directory housing the playbooks and using the **ansible-galaxy collection install -r *collections/requirements.yml** command.
2. When using Ansible Automation Controller, Automation Controller can read the **requirements.yml** file and install collections automatically.
3. It is also possible to install Ansible collections directly from a **TGZ** file or from Ansible Galaxy using the **ansible-galaxy** command on the command line.

It should be noted that the preferred method is leveraging a **requirements.yml** file.

Configuring Ansible to Use Collections

The **ansible.cfg** file controls how Ansible behaves and the basic runtime configuration options. The **[defaults]** section specifically needs to be updated so that Ansible can make use of Ansible collections. It is a common practice to install Ansible Collections in the current working directory of a project. In order for Ansible to look for these collections, the path must be specified in the **ansible.cfg** file.

The **collections_paths** directories are separated by a **:** in the **ansible.cfg** file.

```
[student@workstation my_project]$ cat ansible.cfg
[defaults]
remote_user = root
inventory = inventory
collections_paths =
./collections:~/.ansible/collections:/usr/share/ansible/collections: ①

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

① Specifies the general locations for Ansible to look for the installed Ansible Collections and modules

8.2. Installing and Using Ansible Collections

Ansible collections are installed in a variety of ways using a variety of methods. One of the most common ways for Infrastructure-as-Code (IaC) style projects is the use of a **requirements.yml** file where the **ansible-galaxy** command is used to install the collection prior to the running of the playbook. Ansible playbooks are a convenient method to manage system administration and configuration in an Infrastructure as Code format, because Ansible playbooks are text files that can be easily managed within version control. Having only the **requirements.yml** file as part of the project ensures that the most current roles, collections, and modules are available and present when the code is used a second time. This also allows Ansible Controller to install the collection and modules just before the Job Template Workflow has been run.

When leveraging the **requirements.yml** file to install new roles or collections, there is always a possibility that there could be changes to the modules and syntax provided. When using a **requirements.yml** file, it is good practice to test regularly ensuring any changes to collections, modules, and roles don't break playbook functionality.

There are some arguments that can be made to have the collection content with the project so that it is static and "known good" to be tested with the playbooks and this is something that the development/administration team maintaining the Ansible playbooks and repositories must decide.

It is possible to install collections directly from Ansible Galaxy using the installation method described in the instructions.

```
ansible-galaxy collection install freeipa.ansible_freeipa
```

While the above command works, it is considered to be better practice to install the modules, collections, and roles in the current working directory, in a sub-directory called **collections** located on the same level as the playbooks.

```
ansible-galaxy collection install freeipa.ansible_freeipa -p collections/
```

The command above will install the **freeipa.ansible_freeipa** collections in the current working directory in a sub-directory called collections. The **ansible.cfg** file that specified the collections search path will know to look for collections in the newly installed location.

The most common way of installing the Ansible collections is using the **requirements.yml** file.

```
[student@workstation my_project]$ cat requirements.yml
collections:
- name: ansible.posix
```

The above file will look for the **ansible.posix** collection to install. Since no location was specified, it will automatically look to install the collection from Ansible Galaxy.

```
[student@workstation idm-code]$ ansible-galaxy collection install -r
collections/requirements.yml -p collections/ ①
```

① Assumes there is a collection directory in current working directory with a file called **requirements.yml**

It is important to note that the **ansible.cfg** file controls where and how Ansible searches for installed collections. If a collection is installed for use and isn't properly referenced in the **ansible.cfg** Ansible will be unable to find and use modules from the installed collection.

Chapter 9. The Red Hat IdM Ansible Collection

Red Hat provides the **rhel_idm Ansible Collection** that is fully supported with modules, roles, and other components to assist with the automation and management of Red Hat Identity Management Server. This collection is available through the Red Hat Ansible Automation Platform (AAP) subscription at (https://console.redhat.com/ansible/automation-hub/repo/published/redhat/rhel_idm/).

Collections, modules, roles, and playbooks located on Ansible Galaxy are considered **Upstream** content and are not supported by Red Hat. The **freeipa.ansible_freeipa** collection is the upstream collection to manage Red Hat Identity Management Server and documentation is available at (https://github.com/freeipa/ansible-freeipa/). Even for the official **rhel_idm Collection** collection's documentation is in the Github upstream project.

At the time of this writing, **freeipa.ansible_freeipa** latest version is 1.9.2. The collection contains 99 modules and 6 roles. Throughout the course we will be using only a few of them.

Some of those modules are used from the roles in order to automate the deployment IdM servers and clients.

It is possible to get a listing of those modules from the system after they are installed.

```
[student@workstation Testing]$ ls
collections/ansible_collections/freeipa/ansible_freeipa/plugins/modules
./ipaautomember.py                      ./ipahostgroup.py
./ipaselfservice.py
./ipaautomountkey.py                    ./ipahost.py
./ipaserver_enable_ipa.py
./ipaautomountlocation.py               ./ipaidrange.py
./ipaserver_load_cache.py
./ipapermission.py                            ./ipaserver.py
... OUTPUT OMITTED ...

./ipahbacrule.py                        ./ipareplica_setup_otpd.py
./ipatrust.py
./ipahbacsvcgroup.py                    ./ipareplica_test.py
./ipauser.py
./ipahbacsvc.py                         ./iparole.py
./ipavault.py
```

9.1. Installing freeipa.ansible_freeipa Ansible Collection

In our lab we will use the upstream version of the collection due to the fact that we don't have the necessary AAP subscriptions installed in our lab. In most instances, subscribers with AAP entitlements will leverage Ansible Automation Hub to install any supported Ansible content collection that is needed.

Using the upstream collection

To install the upstream collection, provided that your Ansible control node has Internet Access you can utilize any of the methods we described in the previous section, for example by using the `ansible-galaxy` command.

Using the Supported Red Hat collection

The supported Red Hat Satellite collection is available from (https://console.redhat.com/ansible/automation-hub/repo/published/redhat/rhel_idm/). It can be installed similar to the collections from Ansible Galaxy and the Red Hat Hybrid Cloud Console provide the information on how to install the collection.

```
ansible-galaxy collection install rhel_idm
```

The command to install the collection is the same as Ansible Galaxy. In order to properly install collections from Ansible Automation Hub, the **ansible.cfg** file must be modified so that it can login and authenticate to Automation Hub and so that it knows the URL and collection information

> The **ansible-galaxy** command leverages the **ansible.cfg** file just as the **ansible** and **ansibe-playbook** commands. Having a **[galaxy] section in the file provides the configurations and URLs needed for the *ansible-galaxy** command to use authentication and have the URLs to connect.

9.2. Playbook or roles strategy

When it comes to configuring a system such as IdM with Ansible, there are various methodologies that one can follow. One such methodology is to create a playbook of playbooks, where a playbook calls other playbooks responsible for the configuration of various IdM topics. Each playbook in this scenario would call one or two roles to achieve a specific topic such as user and group management.

Another option is to create a playbook that calls multiple and targeted roles. In this approach, we need to use Ansible tags to configure a specific topic.

There are advantages and disadvantages to each option, and it is ultimately a matter of personal preference which one is used, but bear in mind that using small roles that are dedicated to perform a specific task is closer to the reusability target of Ansible Roles.

Both methodologies involve defining variables either under a specific inventory group or in a separate file. The advantage of creating a playbook of playbooks is that it allows for more fine-grained control over the configuration of specific topics, while the general-purpose playbook approach provides a more streamlined approach.

However, regardless of the chosen methodology,using Ansible tags we ensure that only the tasks we want are executed if needed. This can greatly speed up the configuration process and avoid unnecessary changes. Ultimately, the choice of methodology depends on the specific needs and preferences of the user, and both approaches can be effective for configuring IdM with Ansible.

In the following sections we will use the second approach. As one can read in the Automation structures, or how to name your playbooks, an IdM configuration playbook could be named as `type__rhidm__configure.yml`

```
[student@workstation ~]$ cd  ~/workshop/idm-code
[student@workstation ~]$ cat ./type__rhidm__configure.yml
---
- name: Configure the RH IDM domain
  hosts: "{{ __config_host | groups['rhidms_master'][0] | default('localhost') }}" ①
  roles:
    - rhidm_configure

...
```

① If the controller node is an IdM client, we can use it node for the configuration of IdM, otherwise we need to define another IdM client or server.

9.3. Directory structure for the Ansible code

The directory structure we will use for our automation includes two main directories:

collections

It is used to store reusable modules, plugins, and other content that can be shared across multiple playbooks. Collections can be used to organize and package playbooks, roles, and other content into a single, easily distributable package. Ansible Galaxy, the official Ansible community hub, provides a large collection of pre-built content that can be used in your own playbooks. In addition to Ansible Galaxy, Red Hat Automation Hub is another hub for sharing and distributing Ansible content. Red Hat Automation Hub is a curated collection of automation content provided by Red Hat and its partners. This includes pre-built Ansible roles, modules, and playbooks that are designed to work with Red Hat products and technologies.

roles

It is used to organize playbooks into reusable units called **roles**. A role is essentially a collection of tasks, files, templates, and other content that can be applied to multiple hosts or groups of hosts. By organizing your playbooks into roles, you can make them more modular and easier to maintain. Roles can also be shared across playbooks, making it easy to reuse common functionality across different projects. Overall, the directory structure we use is designed to promote modularity and reusability, making it easier to manage large and complex infrastructure.

Thus, a directory structure like the following is an excellent starting poing.

Listing 4. Initial directory structure for Ansible code

```
(ansible2.9)[student@workstation ~]$
tree
```

```
.
├──── collections
└──── roles
```

Configuring global IdM settings using Ansible playbooks

Ansible roles and modules for FreeIPA

ansible-freeipa: Config Module

Automation structures, or how to name your playbooks

Guided Exercise: Use the freeipa.ansible_freeipa Ansible Collection and configure IdM

Chapter 10. Outcomes

You should be able to

- Clone the repository that contain the Automation Code
- Install the **freeipa.ansible_freeipa** collection
- Configure global IdM configuration as:
    - New users login shell will be the /bin/bash
    - `/bin/bash` is the default shell for the new users.
    - The default e-mail domain is `example.net`
    - The default domain resolution order to `lab.example.net`, `example.net`
- Configure the Global password policy so as new passwords will confront to the following requirements:
    - Be at least 7 characters long
    - Contain at least **3** character classes
    - **Cannot** be changed in less than **24** hours
    - **Must** change every **90** days
    - Must not match the **last 10** password
- In addition, ensure that users can enter up to **3** times a wrong password. Failing to provide a valid password will lock the account for **3 minutes**, and they have to wait for **5 minutes** until the failure counter is reset.

10.1. Prerequisites

You have successfully completed the Guided Exercise "*Prepare your Ansible controller node*"

Opening two tabs in the Gnome Terminal can help you to easily switch between the `idm-inventory` and `idm-code` directories.

Chapter 11. Instructions

1) Login to your workstation as **student** user and open a new terminal. Ensure that you use the `ansible2.9` Python virtual environment.

```
[student@workstation ~]$ source ~/venvs/ansible2.9/bin/activate
(ansible2.9)[student@workstation ~]$
```

2) Navigate to the `workshop/idm-code` directory.

```
(ansible2.9)[student@workstation ~]$ cd ~/workshop/idm-code
```

3) **Clone** the workshop's repository from Github and create a new branch to work on it.

```
(ansible2.9)[student@workstation venvs]$ cd ~/workshop/idm-code
(ansible2.9)[student@workstation ~]$ git clone https://github.com/p-tselios/idm-
wshop.git ./
(ansible2.9)[student@workstation ~]$ git checkout -b <yourname>_workshop
```

Replace the <yourname> with your ROL username without spaces or dots.

4) **Update** the Ansible configuration file so as:

- The collections path includes the `./collections` directory
- The remote user is set to `root`

```
(ansible2.9)[student@workstation idm-code]$ vim ./ansible.cfg
[defaults]
inventory = inventory
remote_user = root
collections_paths = ./collections:/usr/share/ansible/collections

...output ommitted...
```

5) Ensure that the directory `collection` exists.

```
(ansible2.9)[student@workstation idm-code]$ mkdir ./collections
```

6) **Install** the `freeipa.ansible_freeipa` using the `ansible-galaxy` command

```
(ansible2.9)[student@workstation ~]$ cd ~/workshop/idm-code
(ansible2.9)[student@workstation venvs]$ ansible-galaxy collection install \
  freeipa.ansible_freeipa -p ./collections
```

```
[WARNING]: The specified collections path '/home/student/workshop/idm-
code/collections' is not part of the configured
Ansible collections paths
'/home/student/.ansible/collections:/usr/share/ansible/collections'. The installed
collection won't be picked up in an Ansible run. ①

Process install dependency map
Starting collection install process
Installing 'freeipa.ansible_freeipa:1.9.2' to '/home/student/workshop/idm-
code/collections/ansible_collections/freeipa/ansible_freeipa'
```

① You can safely ignore this warning, we will update our Ansible configuration when we clone the repository.

It is possible to see the following error:

ERROR! Unexpected Exception, this is probably a bug: <urlopen error [Errno -2] Name or service not known>

This is known issue in our lab and it’s related to its custom nature. If you receive this error message, please wait 1-2 before you re-try the command.

7) Verify that the **ansible.cfg** configuration for the remote user and the privileges escallation is correct. To do so, use the `setup` module with the `ansible` command.

```
(ansible2.9)[student@workstation idm-code]$ ansible -i ../idm-inventory/ -m setup -b
all
replica1.lab.example.net | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "172.25.250.9"
        ],

...output ommitted...

        "ansible_user_shell": "/bin/bash",
        "ansible_user_uid": 0,
        "ansible_userspace_architecture": "x86_64",
        "ansible_userspace_bits": "64",
        "ansible_virtualization_role": "guest",
        "ansible_virtualization_type": "openstack",
        "discovered_interpreter_python": "/usr/bin/python",
        "gather_subset": [
            "all"
        ],
        "module_setup": true
    },
    "changed": false
}
```

8) Create a **vault** file to store the passwords

```
(ansible2.9)[student@workstation idm-code]$ cd ../idm-inventory
(ansible2.9)[student@workstation idm-inventory]$ ansible-vault create
./group_vars/rhidms/rhidm_secrets.yml
New Vault password: redhat
Confirm New Vault password: redhat
```

```
---
rhidm_admin_password: !unsafe 'RedHat123^'
rhidm_ldap_user_password: !unsafe 'password123!'
...
```

9) Store the variables that are common to all hosts in the inventory under the `group_vars/all` directory in a file with representative name. The variables store information about:

- The domain (**lab.example.net**)

```
(ansible2.9)[student@workstation idm-inventory]$ *mkdir ./group_vars/all*
(ansible2.9)[student@workstation rhidms]$ *vim ./group_vars/all/rhidm_common_vars.yml*
```

```
---
rhidm_domain_name: "lab.example.net"
...
```

11) Define the required variables

11.1) Create the IdM **global configuration** variables file that will configure IdM so as:

- `/bin/bash` is the default shell for the new users.
- The default e-mail domain is `example.net`
- The default domain resolution order to `lab.example.net`, `example.net`
- The Global password policy defines:
  - At least 7 characters long password
  - At least **3** character classes are used
  - **Cannot** be changed in less than **24** hours
  - **Must** change every **90** days
  - Must not match the **last 10** password
- Users can enter up to **3** times a wrong password. Failing to provide a valid password will lock the account for **3 minutes**, and they have to wait for **5 minutes** until the failure counter is reset.

Variables should have a prefix and they should be stored in a file with descriptive name.

```
(ansible2.9)[student@workstation idm-inventory]$ cd ~/workshop/idm-
inventory/group_vars/rhidms
(ansible2.9)[student@workstation rhidms]$ vim ./rhidms_global_configuration.yml
```

```
---

# Default shell for new users
rhidm_defaultshell: "/bin/bash"

# Default email domain for the new users
rhidm_emaildomain: "example.net"

# Set list of domains used for short name qualification
rhidm_domain_resolution_order:
  - lab.example.net
  - example.net

rhidm_default_pw_policy: ①
  minlength: 8
  minclasses: 3
  minlife: 24
  maxlife: 90
  history: 10
  lockouttime: 180
  maxfail: 3
  failinterval: 120
...
```

① The default password policy is defined as a dictionary, since this structure allows us to store information about objects. The password policy is a object with specific options

11.1) **Validate** the variables (Optional)

```
(ansible2.9)[student@workstation rhidms]$ cd ~/workshop/idm-code
(ansible2.9)[student@workstation idm-code]$ ansible -i ../idm-inventory -m debug \
  -a var=rhidm_defaultshell all --ask-vault-pass
Vault password: redhat
idm.lab.example.net | SUCCESS => {
    "rhidm_defaultshell": "/bin/bash"
}
replica2.lab.example.net | SUCCESS => {
    "rhidm_defaultshell": "/bin/bash"
}
replica1.lab.example.net | SUCCESS => {
    "rhidm_defaultshell": "/bin/bash"
}
```

12) Explore the Ansible playbook and roles.

12.1) Navigate to the `idm-code` directory and examine the **type__rhidm__configure.yml** playbook. Verify that it's executed on the `idm.lab.example.net` server.

```
(ansible2.9)[student@workstation idm-inventory]$ cat ./typerhidmconfigure.yml

---
- name: Configure the RH IDM domain
  hosts: "{{ __config_host | default(groups['rhidms_master'][0]) |
default('localhost') }}"
  roles:
    - rhidm_configure

...
```

12.2) Navigate to `roles/rhidm_configure` directory. View its contents.

```
(ansible2.9)[student@workstation rhidm_configure]$ ls -l
total 16
drwxrwxr-x. 2 student student   22 Mar  4 00:49 defaults
drwxrwxr-x. 2 student student   22 Mar  3 23:55 meta
-rwxrwxr-x. 1 student student 6188 Mar  3 23:55 README.md
drwxrwxr-x. 2 student student 4096 Mar  4 00:52 tasks
drwxrwxr-x. 2 student student 4096 Mar  4 01:02 templates
```

12.3) Examing the tasks of the role.

```
(ansible2.9)[student@workstation rhidm_configure]$ cat ./tasks/main.yml
---
- name: Verify that required parameters are set ①
  ansible.builtin.assert:
    that:
      - rhidm_admin_principal is defined
      - rhidm_admin_password is defined
      - rhidm_ds_password is defined
      - rhidm_domain_name is defined
      - rhidm_ldap_read_user is defined
      - rhidm_ldap_user_password is defined

- name: Configure default parameters for users ②
  freeipa.ansible_freeipa.ipaconfig:
    ipaadmin_principal: "{{ rhidm_admin_principal }}"
    ipaadmin_password: "{{ rhidm_admin_password }}"
    defaultshell: "{{ rhidm_defaultshell | default(omit) }}"
    defaultgroup: "{{ rhidm_defaultgroup | default(omit) }}"
    emaildomain: "{{ rhidm_emaildomain | default(omit) }}"
    searchtimelimit: "{{ rhidm_searchtimelimit | default(omit) }}"
    searchrecordslimit: "{{ rhidm_searchrecordslimit | default(omit) }}"
    groupsearch: "{{ rhidm_groupsearch | default(omit) }}"
    domain_resolution_order: "{{ rhidm_domain_resolution_order | default(omit) }}"
  tags:
    - rhidm_config
    - rhidm_config_defaults

- name: Create the default password policy ③
  become: true
  freeipa.ansible_freeipa.ipapwpolicy:
    ipaadmin_principal: "{{ rhidm_admin_principal }}"
    ipaadmin_password: "{{ rhidm_admin_password }}"
    minlength: "{{ rhidm_default_pw_policy.minlength | default(omit) }}"
    minclasses: "{{ rhidm_default_pw_policy.minclasses | default(omit) }}"
    minlife: "{{ rhidm_default_pw_policy.minlife | default(omit) }}"
    maxlife: "{{ rhidm_default_pw_policy.maxlife | default(omit) }}"
    history: "{{ rhidm_default_pw_policy.history | default(omit) }}"
    priority: "{{ rhidm_default_pw_policy.priority | default(omit) }}"
    lockouttime: "{{ rhidm_default_pw_policy.lockouttime | default(omit) }}"
    maxfail: "{{ rhidm_default_pw_policy.maxfail | default(omit) }}"
    failinterval: "{{ rhidm_default_pw_policy.failinterval | default(omit) }}"
    state: "{{ rhidm_default_pw_policy.state | default(omit) }}"
  tags:
    - rhidm_config
    - rhidm_config_defaults
    - rhidm_config_defaults_pw_policy

- name: Create LDAP bind user for Applications
  become: true
  community.general.ldap_entry:
    bind_dn: "cn=Directory Manager"
    bind_pw: "{{ rhidm_ds_password }}"
    start_tls: true
    server_uri: "ldap://localhost/"
    dn: "uid={{ rhidm_ldap_read_user }},cn=sysaccounts,cn=etc,{{'dc=' +
rhidm_domain_name | regex_replace('\\.', ',dc=') }}"
    objectClass:
      - account
      - simplesecurityobject
    attributes:
      uid: "system"
      description: "An LDAP user to allow external applications to authenticate users"
      passwordExpirationTime: "21000101000000Z"
      userPassword: "{{ rhidm_ldap_user_password }}"
    state: present
    validate_certs: "{{ rhidm_validate_certs | default(true) }}"
  no_log: false
  tags:
    - rhidm_config
    - rhidm_config_defaults
    - rhidm_config_defaults_ldap_user
...
```

① Every role requires some variables. Typically we define them in the `defaults/main.yml`. However, in IdM we want to ensure that we use the appropriate domain, the correct passwords etc. Thus we define them in the inventory. This task ensures that the absolute minimum variables are defined.

② This task ensures that new users defaults are specified according to our requirements.

③ This task defines the default password policy

13) Execute the IdM configuration playbook

13.1) Execute the IdM configuration playbook once to apply the desired configuration state

```
[student@workstation ~]$ cd  /home/student/workshop/idm-code
[student@workstation ~]$ cat ./type__rhidm__configure.yml

(ansible2.9)[student@workstation idm-code]$ ansible-playbook --ask-vault-pass \
 -i ../idm-inventory ./type__rhidm__configure.yml

...output ommitted...

PLAY RECAP
*************************************************************************************
*********************************
idm.lab.example.net        : ok=8    changed=2    unreachable=0    failed=0
skipped=2    rescued=0    ignored=0
```

13.2) Check the idempotency of the playbook and roles by executing the playbook again. Verify that the number of tasks reported as **changed** is **0**

```
(ansible2.9)[student@workstation idm-code]$ ansible-playbook --ask-vault-pass \
 -i ../idm-inventory ./type__rhidm__configure.yml

...output ommitted...

PLAY RECAP
*************************************************************************************
*********************************
idm.lab.example.net        : ok=8    changed=0    unreachable=0    failed=0
skipped=2    rescued=0    ignored=0
```

This concludes the section.

Manage IdM Host Groups and Host Automember with Ansible

Chapter 12. Objectives

After completing this section, you should be able to:

- Understand the different strategies regarding the management of IdM users and groups with Ansible
- Understand how to write your inventory files for this

12.1. Managing uses and groups (Overview)

The `freeipa.ansible_freeipa` collection provides a comprehensive range of modules regarding the management of users and user groups.
What distinguishes these modules is their adherence to the same logical structure that underpins the command-line interface of IdM. Specifically, modules dealing with users are dedicated to user-related data exclusively, while modules relating to groups deal with the administration of user groups, including user membership.

This division is a crucial consideration when working with them, particularly when overseeing user group memberships. It is essential to first create individual user accounts, followed by the corresponding user groups, before finally managing user memberships via the appropriate modules.

12.2. Managing Users

Following the approach we discussed previously, the documentation page of the `ipauser` module (https://github.com/freeipa/ansible-freeipa/blob/master/README-user.md), provides us with all the available options of it.

Although this page offers numerous examples, we can appreciate how the module's usage of a dictionary structure for user attributes makes it very flexible. Furthermore, the module offers various alternative strategies for managing users.

Use standard Ansible loops

This strategy requires is the simpler one. We define the user attributes in a variable with a similar structure as the one in the module. The state of each user defines what the module will do for each one of them. An example structure for a task based on this strategy is this:

```
- name: Manage IdM users
  freeipa.ansible_freeipa.ipauser:
    ipaadmin_principal: "{{ rhidm_admin_principal }}"
    ipaadmin_password: "{{ rhidm_admin_password }}"
    name: "{{ item.first[0] }}{{ item.surname }}"
    first: "{{ item.first }}"
    last: "{{ item.surname }}"
    fullname: "{{ item.fullname | default(omit) }}"
    displayname: "{{ item.displayname | default(omit) }}"
    shell: "{{ item.shell | default(omit) }}"
    email: "{{ item.mail | default(omit) }}"
    passwordexpiration: "{{ item.krbpasswordexpiration | default(omit) }}"
    password: "{{ item.password | default(omit) }}"
    random: "{{ true of item.random | bool == true and item.password | default('') |
length == 0 else false }}"
    uid: "{{ item.uid | default(omit) }}"
    gid: "{{ item.gid | default(omit) }}"
    initials: "{{ item.initials | default(omit) }}"
    city: "{{ item.city | default(omit) }}"
    update_password: "on_create"
    userstate: "{{ item.state }}"
    postalcode: "{{ item.zip | default(omit) }}"
    phone: "{{ item.phone | default(omit) }}"
    mobile: "{{ item.mobile | default(omit) }}"
    title: "{{ item.title | default(omit) }}"
    manager: "{{ item.manager | default(omit) }}"
    userauthtype: "{{ item.userauthtype | default(omit) }}"
    userclass: "{{ item.userclass | default(omit) }}"
    certificate: "{{ item.userclass | default(omit) }}"
    sshpubkey: "{{ item.sshpubkey | default(omit) }}"
    noprivate: "{{ item.sshpubkey | default(false) }}"
  loop: "{{ rhidm_users }}"
  loop_control:
    label: "{{ item.first }} {{ item.last }} - "{{ item.state | default('present')
}}""
```

The structure of the module, dictates the structure of the variables. It’s clear that we will need either a list of dictionaries or a dictionary of dictionaries. There are arguments supporting either case and is merely a decision of the devops engineers which can be taken only when they take into account the architecure of the IdM deployment as a whole.

For the purposes of this course, we will create a list of dictionaries.

```
rhidm_users:
  - first: 'John'
    last: 'Doe'
    password: 'Passw0rd!'
    state: present
  - first: "Peter"
    last: "Smith"
    password: !unsafe 'Secret123!'
  - first: "Marie"
    last: "Johnson"
    password: !unsafe 'TopSecret!'
    state: absent
```

An important drawback of this methodology is that Ansible will iterate over each user in the list, and for each one of them, it will make an API call to the IdM server. This could result in significant delays, especially when configuring multiple users. As a result, it is critical to consider this aspect when designing our solution and implement appropriate measures to mitigate any performance issues.

Distinguish the user actions per task

An alternative approach to manage users with our Ansible role is to separate the actions into different tasks. We can use the ipauser module’s six different states (`present`, `absent`, `enabled`, `disabled`, `unlocked`, and undeleted) to generate lists of dictionaries with users belonging to each category dynamically. This approach can be more complex during development and testing, but once the variables are set up correctly, it can significantly reduce the execution time of the playbook.

An example task could be like the following:

```
- name: Generate the list of users that we will create
  ansible.builtin.set_fact:
    __dynamic_list_for_create_users: "{{ rhidm_users |
ansible.builtin.selectattr('state', 'undefined')
    | union(rhidm_users | ansible.builtin.selectattr('state', "equalto", 'present') |
list }}"

- name: Generate the list of users that we will disable
  ansible.builtin.set_fact:
    __dynamic_list_for_disabled_users: "{{ rhidm_users |
ansible.builtin.selectattr('state', 'defined')
    | union(rhidm_users | ansible.builtin.selectattr('state', "equalto", 'disabled') |
list }}"

- name: Create IdM users
  freeipa.ansible_freeipa.ipauser:
    ipaadmin_principal: "{{ rhidm_admin_principal }}"
    ipaadmin_password: "{{ rhidm_admin_password }}"
    users: "{{ __dynamic_list_for_create_users }}"
    state: present
    update_password: on_create

- name: Disable IdM users
  freeipa.ansible_freeipa.ipauser:
    ipaadmin_principal: "{{ rhidm_admin_principal }}"
    ipaadmin_password: "{{ rhidm_admin_password }}"
    users: "{{ __dynamic_list_for_disabled_users }}"
    state: disabled
```

12.3. Managing Groups

Managing groups in IDM using the `freeipa.ansible_freeipa` module is more involved than managing users. The group module enables us to ensure the presence and absence of groups and their members. The module’s documentation is available at https://github.com/freeipa/ansible-freeipa/blob/master/README-group.md.

The module has a dual functionality. The first is to manage groups by creating or deleting them, while the second is to manage group memberships. As mentioned before, we must first create the user and user groups and then manage the group memberships.

In addition to standard POSIX groups, the module also allows us to create non-POSIX and external groups. These groups are particularly useful when establishing a trust relationship with Active Directory.

Building the group-related variables

When managing group membership with freeipa.ansible_freeipa, one of the decisions that a DevOps engineer needs to make is how to represent a user's group membership in the inventory variables. There are two options available to choose from.

The first option is to include the group membership variables as part of the user's data. This approach is very efficient when it comes to administering the relevant variables. However, the disadvantage of this approach is that it generates a relatively flat structure, which might not be suitable for more complex setups.

An example of a data structure for this approach is show in the following example.

```
rhidm_users:
  - first: 'John'
    last: 'Doe'
    password: 'Passw0rd!'
    state: present
    groups:
      - group01
      - group02
      - group03
  - first: "Peter"
    last: "Smith"
    password: !unsafe 'Secret123!'
    groups:
      - group01
  - first: "Marie"
    last: "Johnson"
    password: !unsafe 'TopSecret!'
    state: absent
```

A second approach is to define the group membership by including users as members of the group in the group definition. This approach can result in more complex structures, as groups can be included as members of other groups etc. However, this approach can lead to potential data inconsistencies as user data is managed separately from the group membership data.

An example of a data structure for this approach is show in the following example.

```
rhidm_user_groups:
  - name: idmgroup01
```

```
    gid: 50000
    description: >
      IdM group used by Ansible-created users
    users:
      - jdoe
      - psmith
    subgroups:
      - subgroup01
      - subgroup02
    external: true
     externalmember:
       - 'EXAMPLE.NET\aduser01'
       - 'EXAMPLE.NET\aduser02'
```

The decision on the data structures and on which conceptual object, users or groups, we define the group membership determines the way we write the corresponding tasks. Therefore, it is important to carefully consider the pros and cons of each approach before making a decision. For simplicity reasons during this workshop, we will define the group membership of users as part of the group's definition.

To define a group as a member of another group, it's necessary to create the child group **before** the parent group. While the use of nested user groups is possible, it's important to keep this ordering in mind to avoid errors. The behavior of ensuring child groups are created before parent groups is consistent with the behavior of the ipa command line tool.

Writing the Group management task

The group management task is straightforward and easy to implement. The documentation page provides the skeleton for our task.

```
- name: Manage the IdM User groups
  freeipa.ansible_freeipa.ipagroup:
    ipaadmin_principal: "{{ rhidm_admin_principal }}"
    ipaadmin_password: "{{ rhidm_admin_password }}"
    name: "{{ item.name }}"
    description: "{{ item.description }}"
    gid: "{{ item.gid | default(omit) }}"
    posix: "{{ item.external | default(true) }}"
    external: "{{ item.external | default(omit) }}"
    nonposix: "{{ item.external | default(omit) }}"
    externalmember: "{{ item.externalmember | default(omit) }}"
    state: "{{ item.state | default('present') }}"
  loop: "{{ rhidm_user_groups }}"
  loop_control:
    label: "{{ item.name }} - {{ item.state | default('present') }}"
```

This Ansible task can perform the creation or deletion of a group, as evident from the provided YAML structure. However, an exception exists for managing the group membership of "external" groups, such as those consisting of Active Directory users. In this case, the task can also handle the management of group membership by providing a list of external users.

Writing the Group membership task

The group membership task is even more straightforward and easy to implement. Again the documentation page provides the skeleton for our task.

```
- name: Manage the IdM User groups membership
  freeipa.ansible_freeipa.ipagroup:
    ipaadmin_principal: "{{ rhidm_admin_principal }}"
    ipaadmin_password: "{{ rhidm_admin_password }}"
    name: "{{ item.name }}"
    action: member
    user: "{{ item.user | default([]) }}"
    group: "{{ item.group | default([]) }}"
  loop: "{{ rhidm_user_groups }}"
  loop_control:
    label: "{{ item.name }} - {{ item.state | default('present') }}"
```

The task requires the `rhidm_user_groups` variable to have the appropriate structure to support group membership. If we define the group membership in the user's definition, we need to dynamically construct a new variable to hold the group membership information.

One very important consideration is the way that existing group members are handled from this module. Assuming that we have a group with the name `demogroup` and we have already defined users `user01`, `user02`, `user03` as its members.
Modifying our inventory variable so as `user03` is no longer a member of `group01` will **not** result to the removal of the user from it. The task will ensure that users `user01` and `user02` are members of the group and since they already are, it will not perform any additional action!

In general, the behavior of the `freeipa.ansible_freeipa` modules differs from the older community-supported IPA modules, where inventory values were considered as the desired state of the IdM object, and corrective actions were taken in case of deviations.
With the `freeipa.ansible_freeipa module`, inventory values are still considered as the desired state of the IdM object; however, any existing items or attributes defined outside of them are not modified.

Managing the Auto-member rules

The `freeipa.ansible_freeipa` modules handles the management of automember rules for both user groups and host groups in a similar way. As a DevOps engineer, there are two options for creating the necessary data structures in the inventory.

The first option is to create a dedicated data structure for all the automember rules, both for user and host groups. Since hosts are not directly managed from the inventory, a separate data structure is needed. Including group automember rules in this structure is logical.

The second option is to store the automember information within the relevant user and host group data structures. While this approach may make the data structures more complex, it ensures there are no inconsistencies between the groups and the automember rules.

Reading the documentation page for the automember module in https://github.com/freeipa/ansible-freeipa/blob/master/README-automember.md we note all the important variables we need to define.

Prioritizing data consistency, we can choose to update the data structure for the groups as follows:

```
rhidm_user_groups:
  - name: idmgroup01
    gid: 50000
    description: >
      IdM group used by Ansible-created users
    users:
      - jdoe
      - psmith
    subgroups:
      - subgroup01
      - subgroup02

    automember_state: present
    automember_description: A description for the AM rule
    automember_incrules:
      - key: cn
        expr: 'inexp2'
        state: absent
    automember_excrules:
      - key: cn
        expr: 'exexp2'
        state: present
```

The data structure above can use a dictionary to represent the `automember` related information. However this requires more processing prior to the execution.

This concludes the section.

Guided Exercise: Manage users and Groups in IdM

Chapter 13. Outcomes

You should be able to:

- Create, delete and modify users, user groups and external user groups

13.1. Prerequisites

You have successfully completed the previous Guided Exercises.

Opening two tabs in the Gnome Terminal can help you to easily switch between the `idm-inventory` and `idm-code` directories.

Chapter 14. Instructions

1) Login to your workstation as **student** user and open a new terminal. Ensure that you use the `ansible2.9` Python virtual environment.

```
[student@workstation ~]$ source ~/venvs/ansible2.9/bin/activate
(ansible2.9)[student@workstation ~]$
```

2) Navigate to the `workshop/idm-code` directory.

```
(ansible2.9)[student@workstation ~]$ cd ~/workshop/idm-code
```

3) Examing the role `rhidm_manage_users`

3.1) Examine the default values of the role's variables. Note the name and the structure of the variables used by the role.

```
(ansible2.9)[student@workstation ~]$ cat ~/workshop/idm-
code/roles/rhidm_manage_users/defaults/main.yml
```

3.2 Examine the task file to identify the mandatory variables

```
(ansible2.9)[student@workstation ~]$ cat ~/workshop/idm-
code/roles/rhidm_manage_users/tasks/main.yml
```

4) Create a list of dictionaries that will represent the IdM users as defined in the following table:

first name	last name	uid	shell	password	state
Work	Shop01	60000	/bin/sh	5evenof9	present
Work	Shop02	60001	Default shell	b0rgque!	present
Work	Shop03	60002	/bin/ksh	1mper1o	absent
Work	Shop04	60003	/bin/zsh	Sisko99	present
Work	Shop05	60004	Default shell	Data!23	preserved

4.1 Open a new tab in your terminal and create a new file in the IdM inventory that will store the user information.

```
(ansible2.9)[student@workstation ~]$ vim ~/workshop/idm-
inventory/group_vars/rhidms/rhidm_user_data.yml
```

```
---
```

```
rhidm_user_list:
  - first: 'Work'
    last: 'Shop01'
    uid: 60000
    password: !unsafe '5evenof9' #Seven of Nine (Tertiary Adjunct of Unimatrix 01)
    loginshell: "/bin/sh"
    state: present
  - first: "Work"
    last: "Shop02"
    uid: 60001
    password: !unsafe 'b0rgque!' #Locutus of Borg (Jean-Luc Picard)
  - first: "Work"
    last: "Shop03"
    uid: 60002
    password: !unsafe '1mper1o'  #The Borg Queen
    loginshell: "/bin/ksh"
    state: absent
  - first: "Work"
    last: "Shop04"
    uid: 60003
    loginshell: "/bin/zsh"
    password: !unsafe 'Sisko99'  #Capt. Benjamin Sisko
    state: present
  - first: "Work"
    last: "Shop05"
    uid: 60004
    password: !unsafe 'Data!23'  #Lt. Cmdr. Data
    state: preserved
...
```

5) Include the role `rhidm_manage_users` in your playbook and execute it

5.1) Modify the playbook `type__rhidm__configure.yml` to include the role `rhidm_manage_users`

```
[student@workstation ~]$ vim  /home/student/workshop/idm-code/type__rhidm
__configure.yml
---
- name: Configure the RH IDM domain
  hosts: "{{ __config_host | groups['rhidms_master'][0] | default('localhost') }}"
  roles:
    - role: rhidm_configure
      tags: idm_conf

    - role: rhidm_manage_users
      tags: idm_users
...
```

5.2) Execute the playbook to manage users

```
(ansible2.9)[student@workstation ~]$ cd ~/workshop/idm-code/
(ansible2.9)[student@workstation ~]$ ansible-playbook --ask-vault-pass \
 -i ../idm-inventory ./type__rhidm__configure.yml

 ...output omitted...

TASK [rhidm_manage_users : Manage users]
*************************************************************************************
***
changed: [idm.lab.example.net] => (item=WShop01 - present)
changed: [idm.lab.example.net] => (item=WShop02 - present)
ok: [idm.lab.example.net] => (item=WShop03 - absent)
changed: [idm.lab.example.net] => (item=WShop04 - present)
ok: [idm.lab.example.net] => (item=WShop05 - absent)

TASK [rhidm_manage_users : Apply roles to users]
********************************************************************************


PLAY RECAP
*************************************************************************************
*********************************
idm.lab.example.net        : ok=6    changed=1    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0
```

6) Create a list of dictionaries that will represent the IdM groups as defined in the following table:

Group name	GID	Description	POSIX/External	Members
WSGroup01	120000	The first group of the workshop	POSIX	WShop01 WShop04
WSGroup02	N/A	An external group	External	N/A

In addition, group `WSGroup01` should have an **automember** rule so all users for which their username starts with `wsgroup` followed by two numbers between `0` and `2` should be members of this group.

6.1) Open a new tab in your terminal and create a new file in the IdM inventory that will store the group information.

```
(ansible2.9)[student@workstation ~]$ vim ~/workshop/idm-
inventory/group_vars/rhidms/rhidm_group_data.yml
```

```
---
  - name: WSGroup01
    description: >
      The first group of the workshop
    users:
      - wshop01
      - wshop04
    automember_state: present
    automember_description: "Workstop IdM users"
    automember_inclrules:
      - key: cn
        expr: '^wshop0[0-2].*'
        state: present
    state: present

  - name: WSGroup02
    description: "An external group"
    gid: 120000
    external: true
...
```

If the module variable `externalmember` is specified, you need define the `external: true`, otherwise the task will fail.

Please remember that the module variables `external`, `posix`, `nonposix` are mutually exclusive.

Due to a bug in the `freeipa.ansible_freeipa` collection adding external users to an external group is successful on the first execution of a playbook, but subsequent tasks will fail. Thus, we don't add any `externalmember` in the variable.

7) Include the role `rhidm_manage_groups` in your playbook and execute it

5.1) Modify the playbook `type__rhidm__configure.yml` to include the role `rhidm_manage_groups`

```
[student@workstation ~]$ vim  /home/student/workshop/idm-code/type__rhidm
__configure.yml
---
- name: Configure the RH IDM domain
  hosts: "{{ __config_host | groups['rhidms_master'][0] | default('localhost') }}"
  roles:
    - role: rhidm_configure
      tags: idm_conf

    - role: rhidm_manage_users
      tags: idm_users

    - role: rhidm_manage_groups
      tags: idm_groups
...
```

5.2) Execute the playbook to manage users

```
(ansible2.9)[student@workstation ~]$ cd ~/workshop/idm-code/
(ansible2.9)[student@workstation ~]$ ansible-playbook --ask-vault-pass \
 -i ../idm-inventory ./type__rhidm__configure.yml

 ...output ommitted...

TASK [rhidm_manage_groups : Manage the IdM User groups]
**************************************************************************
changed: [idm.lab.example.net] => (item=WSGroup01 - present)
changed: [idm.lab.example.net] => (item=WSGroup02 - present)

TASK [rhidm_manage_groups : Build the automember variables]
**********************************************************************
ok: [idm.lab.example.net]

TASK [rhidm_manage_groups : Create or delete Automember rules for Users]
*********************************************************
changed: [idm.lab.example.net] => (item=WSGroup01 - present)

TASK [rhidm_manage_groups : Add Automember conditions]
***************************************************************************
changed: [idm.lab.example.net] => (item=WSGroup01)

TASK [rhidm_manage_groups : Remove Automember conditions]
************************************************************************


RUNNING HANDLER [rhidm_manage_groups : Rebuild User Group membership]
************************************************************
changed: [idm.lab.example.net]

PLAY RECAP
**************************************************************************************
*********************************
idm.lab.example.net        : ok=6    changed=4    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0
```

This concludes the section.

Manage IdM Host Groups and Host Automember with Ansible

Chapter 15. Objectives

After completing this section, you should be able to:

- Understand the different strategies regarding the management of IdM Host Groups with Ansible
- Understand how to write your inventory files for this

15.1. Managing Host Groups (Overview)

In contrast to user and user group management, host group management requires a different approach. With user and user group management, the focus is on defining access control policies and managing user accounts. However, with host groups, we want to maintain a lot of control over which hosts belong to which group, and at the same time, maximize the use of automember rules to provide flexibility to system administrators and speed up the lifecycle management of IdM clients.

Automember rules allow us to dynamically assign hosts to host groups based on certain criteria, such as host name patterns, IP address ranges, or other attributes. This can greatly simplify the process of managing a large number of hosts and ensure that they are always assigned to the correct host group.

However, it’s important to balance the use of automember rules with the need for control and oversight. We need to ensure that the hosts are properly classified and that access control policies are enforced. For example, we may want to limit access to certain host groups based on security requirements or business policies.

By using Ansible to manage IdM host groups, we can achieve this balance between control and flexibility. Ansible allows us to define playbooks that specify exactly which tasks should be performed on which hosts and in what order. This provides us with fine-grained control over the management of our IdM infrastructure, while still allowing us to take advantage of automember rules to simplify the management of large numbers of hosts.

Using Hostgroups will allow us to simplify the HBAC and SUDO management on each host regardless of the number of them, provided that we have a solid method to distinguih them. For example, if we provision new RHEL systems via Red Hat Satellite server, integrating Red Hat Satellite with Red Hat Identity Management will ensure that every new host is a member of the correct Host Group.As a result of this classification, HBAC and SUDO rules will be applied to the new host automatically, without the need for manual intervention.

By using Ansible to manage IdM host groups, we can automate the management of HBAC and SUDO policies across our entire infrastructure, ensuring that access control policies are consistently applied and reducing the risk of errors and misconfigurations.

The `freeipa.ansible_freeipa` collection provides a comprehensive range of modules regarding the management of hosts and host groups.

15.2. Strategies for managing host-related data in idm deployments

The discussion should make it clear that there is not a single strategy regarding host-related data and how we store it in the inventory. One approach is to store information about hosts, such as description and userclass, in a file with a representative name and store HBAC and automember rules in separate data structures. This can be a viable option for small IdM deployments. However, in larger IdM deployments, the process of updating information about a host in different places can be tedious and error-prone.

In order to keep data structures as simple as possible, we will focus on a single approach in this book. However, readers are invited to experiment with different strategies and be flexible in their approach. For example, an organization may initially store host information under the host_vars/hostname directory, but later determine that it is not the appropriate strategy for their modus operandi. In such cases, it is important to be able to adapt and modify the inventory data structures to meet the changing needs of the organization.

One important consideration when choosing a strategy for handling host-related data is to ensure that it is scalable and easy to maintain. In larger IdM deployments with hundreds or even thousands of hosts, it can be challenging to keep track of host information and ensure that it is accurate and up-to-date. Therefore, it is important to have clear processes and procedures in place for updating host information, as well as for managing HBAC and automember rules.

15.3. Managing Host Groups

Following the approach we discussed previously, the documentation page of the `ipahost` module (https://github.com/freeipa/ansible-freeipa/blob/master/README-host.md), provides us with all the available options of it.

Update host information

If IdM clients are enrolled with a supported method like the `ipa-client-install` command, directly from a Red Hat Satellite server during the provisioning phase, or by using the `freeipa.ansible-freeipa` roles, we don’t have to create new hosts in the IdM manually. Instead, we can update their information, such as their description, SSH public keys, SSL certificates, etc.

As we discussed in previous sections, IdM provides a limited list of attributes for each host. Administrators can use locality, location, or platform to group together hosts with similar attributes. However, these attributes do not provide us with the granularity needed for more specific groupings. For example, there are no attributes that could be used to identify VMs hosting MySQL 10 in the Production environment.

In a Red Hat Satellite server, administrators use host groups to perform fine-tuned grouping of servers like the above. In IdM, this information is stored in the **userclass** attribute, which can be utilized by automember rules to classify a server to the appropriate host group. By utilizing the userclass attribute in conjunction with automember rules, we can classify hosts based on specific attributes such as the type of application running on the host, the version of the software, or any other custom attribute that we define. This provides us with the flexibility needed to manage our infrastructure effectively and efficiently.

If we don’t utilize Red Hat Satellite server during the provisioning phase, typically information such as the description of a host or the userclass value is defined as an inventory host or group variable. In the case where we want to add multiple attributes to the userclass parameter, we need to use expressions that are friendly to a regular expression since the latter is used in the automember rules. For example, we could define a userclass value for a group of hosts as "production-webserver-mysql7", where "production" represents the environment, "webserver" represents the application type, and "mysql7" represents the version of the software running on the host. This way, we can use regular expressions in the automember rules to classify hosts based on these attributes.

```
host_description: "Mysql 10 - Production server"
host_userclass: "app=MysqlSrv|srvenv=prod"
```

Host group data

For the purposes of this book we will keep the data structures for host groups as simple as possible, in expense of possible data inconsistencies.

```
rhidm_host_groups_list:
  - name:  idmhostgroup
    description: >
      Database Servers in the production environment
    state: present
    automember: yes
    incrules:
      - key: userclass
        expression: "env=prod;app=mysql;deployment_env=prod;[;|~]"
```

Writing the Host management task

The host management task is straightforward and easy to implement. The documentation page (https://github.com/freeipa/ansible-freeipa/blob/master/README-host.md) provides the skeleton for our task.

```
  - name: Update host data
    freeipa.ansible_freeipa.ipahost:
      ipaadmin_principal: "{{ rhidm_host_admin_principal }}"
      ipaadmin_password: "{{ rhidm_admin_password }}"
      name: "{{ inventory_hostname }}"
      ip_address: "{{ ansible_all_ipv4_addresses + ansible_all_ipv6_addresses if
rhidm_has_dns | default(false) == true else omit }}"
      update_dns: "{{ rhidm_has_dns | default(false) }}"
      userclass: "{{ host_userclass }}"
      state: present
  tags:
    - update_hostdata
```

Host Automember rules

The `freeipa.ansible_freeipa` module we used for managing user group Automember rules can also be used for managing host group Automember rules. In this book, we keep the management of Automember rules specific to their targets. For instance, the Automember rules for user groups are handled by the rhidm_user_groups role. Similarly, we will manage the host group Automember rules from the rhidm_host_groups role.

Another approach is to create a separate role that manages all the Automember rules. There are no clear advantages or disadvantages to each method, so organizations can choose the approach that suits them best.

Management of host groups automembership is similar to the relevant automember rules for users, as discussed in the in the Managing the Auto-member rules section.

Writing Host Automember rules

The host automember rules, provided that we follow the approach of separate data structures is quite simple. An example task is the following:

```
- name: Manage Host Group Automembers
  freeipa.ansible_freeipa.ipaautomember:
    ipaadmin_password: "{{ rhidm_admin_password }}"
    name: "{{ item.name }}"
    description: "{{ item.description | default(omit) | trim }}"
    automember_type: hostgroup
    state: "{{ item.state | default('present') }}"
    inclusive: "{{ item.increles | default(omit) }}"
    exclusive: "{{ item.excrules | default(omit) }}"
  loop: "{{ rhidm_hostgroups | json_query('[?automember==`true`]') | list }}"
  loop_control:
    label: "{{ item.name }}"
```

One critical aspect is that devops engineers should handle the management of host groups in tandem. One cannot seperate the automember rules from the host data or the host group information. It is also important to note that some form or data transformation is needed due to the requirements of the `freeipa.ansible_freeipa.ipaautomember` module.

Writing HBAC Rules

As discussed already, HBAC rules control user access on the IdM clients. As a minimum a devops engineer provides a name, description a host or hostgroup on which the rule is applied, the users or groups that are affected from this rule and optionally a service.

The documentation of the relevant module (https://github.com/freeipa/ansible-freeipa/blob/master/README-hbacrule.md) verifies that calling the module is straightforward.

A typical data structure of the HBAC rules in the inventory would resemple the following example.

```
rhidm_hbac_rules:
  - name: allow_all
    state: disabled

  - name: hbr_01
    users:
      - aba01
    hosts:
      - idmc7.home
    services: ①
      - ftp
```

① Services are not extensively used. Typically, services are considered optional and are handled as such in the inventory data structures.

Applying HBAC rules with Ansible

HBAC rules management with the `freeipa.ansible_freeipa.ipahbacrule` module follows the same philosophy as other modules where we need to define the rule in 2 distinctive steps.

```
- name: Manage HBAC rules
  freeipa.ansible_freeipa.ipahbacrule:
    ipaadmin_password: "{{ rhidm_admin_password }}"
    name: "{{ item.name }}"
    description: "{{ item.description | default(omit) | trim }}"
    usercategory: "{{ item.usercategory | default('all') if item.usercategory is
undefined and item.state | default('present') != 'absent' and item.state |
default('present') != 'disabled' else omit }}"
    hostcategory: "{{ item.hostcategory | default('all') if item.hostcategory is
undefined and item.state | default('present') != 'absent' and item.state |
default('present') != 'disabled' else omit }}"
    servicecategory: "{{ item.servicecategory | default('all') if item.service is
undefined and item.state | default('present') != 'absent' and item.state |
default('present') != 'disabled' else omit }}"
    host: "{{ item.host | default(omit) }}"
    hostgroup: "{{ item.hostgroup | default(omit) }}"
    hbacsvc: "{{ item.service | default(omit) }}"
    hbacsvcgroup: "{{ item.servicegroup | default(omit) }}"
    user: "{{ item.user | default(omit) }}"
    group: "{{ item.usergroup | default(omit) }}"
    state: "{{ item.state | default('present') }}"
  loop: "{{ rhidm_hbac_rules }}"
  loop_control:
    label: "{{ item.name }}"
  tags:
    - rhidm_config
    - rhidm_config_hbac

- name: Manage HBAC membership
  freeipa.ansible_freeipa.ipahbacrule:
    ipaadmin_password: "{{ rhidm_admin_password }}"
    name: "{{ item.name }}"
    description: "{{ item.description | default(omit) | trim }}"
    usercategory: "{{ item.usercategory | default('all') if item.usercategory is
undefined and item.state | default('present') != 'absent' and item.state |
default('present') != 'disabled' else omit }}"
    hostcategory: "{{ item.hostcategory | default('all') if item.hostcategory is
undefined and item.state | default('present') != 'absent' and item.state |
default('present') != 'disabled' else omit }}"
    servicecategory: "{{ item.servicecategory | default('all') if item.service is
undefined and item.state | default('present') != 'absent' and item.state |
default('present') != 'disabled' else omit }}"
    host: "{{ item.host | default(omit) }}"
    hostgroup: "{{ item.hostgroup | default(omit) }}"
    hbacsvc: "{{ item.service | default(omit) }}"
    hbacsvcgroup: "{{ item.servicegroup | default(omit) }}"
    user: "{{ item.user | default(omit) }}"
    group: "{{ item.usergroup | default(omit) }}"
    action: member
    state: "{{ item.state | default('present') }}"
  loop: "{{ rhidm_hbac_rules }}"
  loop_control:
    label: "{{ item.name }}"
  tags:
    - rhidm_config
    - rhidm_config_hbac
```

The task requires the `rhidm_hbac_rules` variable to have the appropriate structure to support group membership. If we define the group membership in the user's definition, we need to dynamically construct a new variable to hold the group membership information.

This concludes the section.

Guided Exercise: Manage hosts and Host groups in IdM

Chapter 16. Outcomes

You should be able to:

- Create, delete and modify host groups
- Manage Automember rules for Host Groups
- Create HBAC Rules for host groups

16.1. Prerequisites

You have successfully completed the previous Guided Exercises.

Opening two tabs in the Gnome Terminal can help you to easily switch between the `idm-inventory` and `idm-code` directories.

Chapter 17. Instructions

1) Login to your workstation as **student** user and open a new terminal. Ensure that you use the `ansible2.9` Python virtual environment.

```
[student@workstation ~]$ source ~/venvs/ansible2.9/bin/activate
(ansible2.9)[student@workstation ~]$
```

2) Navigate to the `workshop/idm-code` directory.

```
(ansible2.9)[student@workstation ~]$ cd ~/workshop/idm-code
```

3) Examing the role `rhidm_manage_hosts`

3.1) Examine the default values of the role's variables. Note the name and the structure of the variables used by the role.

```
(ansible2.9)[student@workstation ~]$ cat ~/workshop/idm-
code/roles/rhidm_manage_hosts/defaults/main.yml
```

3.2 Examine the task file to identify the mandatory variables

```
(ansible2.9)[student@workstation ~]$ cat ~/workshop/idm-
code/roles/rhidm_manage_hosts/tasks/main.yml
```

4) Update the host information of `client.lab.example.net` to

- Have the description "Workstop client"
- Containt the env=prod;app=mysql;deployment_env=prod; in its userclass attribute Have the description

4.1) Create a directory under the `host_vars` if needed in the inventory.

```
(ansible2.9)[student@workstation ~]$ mkdir -p ~/workshop/idm-
inventory/host_vars/client.lab.example.net
```

4.2) Create a variable file with the name `host_data.yml` with the following variables defined:

```
(ansible2.9)[student@workstation ~]$ vim ~/workshop/idm-
inventory/host_vars/client.lab.example.net/host_info.yml
```

```
---
```

```
host_description: "Mysql - Production server"
host_userclass: 'env=prod;app=mysql;deployment_env=prod'
...
```

4.3) Update the inventory to include the IdM clients group and hosts

```
(ansible2.9)[student@workstation ~]$ *vim ~/workshop/idm-inventory/inventory
 ...output omitted...

[rhidm_clients]
client.lab.example.net
```

5) Create a list of dictionaries that will represent the IdM host groups as defined in the following table:

Group Name	Description	Hosts	Automember Rule	Include Rule
wsclients	Workstop clients	client.lab.example.net	Yes	env=prod;app=mysql;deployment_env=prod

5.1 Create a new file in the IdM inventory that will store the host group information.

```
(ansible2.9)[student@workstation ~]$ vim ~/workshop/idm-
inventory/group_vars/rhidms/rhidm_hgroup_data.yml
```

```
---
rhidm_hostgroups:
  - name: "wsclients"
    description: "Workstop clients"
    automember: true
    automember_incrules:
      - key: userclass
        expr: 'env=prod;app=mysql;deployment_env=prod'
...
```

1. Create a list of dictionaries that will represent the IdM HBAC rules as defined in the following table:

HBAC Name	State	Hosts	Host Groups	Users	Groups	Services
all_all	Disabled	N/A	N/A	N/A	N/A	N/A
allow_client	Enabled	N/A	wsclients	N/A	wsgroup01	All

The `allow_client` rule should allow only users that are members of the `wsgroup01` to access hosts that belong to the host group wsclients to connect to it. All other users should not be able to login to the host.

6.1) Create a new file in the IdM inventory that will store the HBAC information.

```
(ansible2.9)[student@workstation ~]$ vim ~/workshop/idm-
inventory/group_vars/rhidms/rhidm_hbac.yml
```

```
---
rhidm_hbac_rules:
  - name: allow_all
    state: disabled

  - name: allow_client
    hostgroup: wsclients
    group: wsgroup01
...
```

7) Include the roles rhidm_manage_hosts and rhidm_manage_hbac in your playbook and execute it

7.1) Modify the playbook type__rhidm__configure.yml to include the role rhidm_manage_users

```
[student@workstation ~]$ vim  /home/student/workshop/idm-code/type__rhidm
__configure.yml
---
- name: Configure the RH IDM domain
  hosts: "{{ __config_host | groups['rhidms_master'][0] | default('localhost') }}"
  roles:
    - role: rhidm_configure
      tags: idm_conf

    - role: rhidm_manage_users
      tags: idm_users

    - role: rhidm_manage_groups
      tags: idm_groups

    - role: rhidm_manage_hosts
      tags: idm_hosts

    - role: rhidm_manage_hbac
      tags: idm_hbac
...
```

7.2) Execute the playbook to manage hosts

```
(ansible2.9)[student@workstation ~]$ cd ~/workshop/idm-code/
```

```
(ansible2.9)[student@workstation ~]$ ansible-playbook --ask-vault-pass \
 -i ../idm-inventory ./type__rhidm__configure.yml

 ...output ommitted...

TASK [rhidm_manage_hosts : Update host data]
*************************************************************************************
changed: [idm.lab.example.net] => (item=client.lab.example.net)

TASK [rhidm_manage_hosts : Manage the host groups]
*******************************************************************************
changed: [idm.lab.example.net] => (item=wsclients - present)

TASK [rhidm_manage_hosts : Build the automember variables]
***********************************************************************
ok: [idm.lab.example.net]

TASK [rhidm_manage_hosts : Create or delete Automember rules for Hosts]
**********************************************************
changed: [idm.lab.example.net] => (item=wsclients - present)

...output ommitted...
TASK [rhidm_manage_hbac : Manage HBAC rules]
*************************************************************************************
changed: [idm.lab.example.net] => (item=allow_all)
changed: [idm.lab.example.net] => (item=allow_client)

TASK [rhidm_manage_hbac : Manage HBAC membership]
********************************************************************************
ok: [idm.lab.example.net] => (item=allow_client)

PLAY RECAP
**************************************************************************************
*********************************
idm.lab.example.net        : ok=19   changed=6    unreachable=0    failed=0
skipped=3    rescued=0    ignored=0
```

8) Login to idm server, obtain a Kerberos ticket for the admin user and verify the above actions

```
(ansible2.9)[student@workstation ~]$ ssh idm
[student@idm ~]$ kinit admin
Password for admin@LAB.EXAMPLE.NET: RedHat123^

[student@idm ~]$ ipa host-show client.lab.example.net
  Host name: client.lab.example.net
  Description: Mysql
  Principal name: host/client.lab.example.net@LAB.EXAMPLE.NET
  Principal alias: host/client.lab.example.net@LAB.EXAMPLE.NET
  SSH public key fingerprint: SHA256:SKPbFqfD9FC9OOBraDzDLrXjAqfmRPYl99C7Cn3zak0 (ssh-
rsa), SHA256:lXmpBW0vCfgwrSGzb0qS9KzIlg2fTmp8k/lIsuCW21Q
```

```
                              (ecdsa-sha2-nistp256),
SHA256:lfaB68WElm4Pg78ryMo6yNL3cYxZmpE4dH10s2LhgRU (ssh-ed25519)
  Class: env=prod;app=mysql;deployment_env=prod
  Assigned ID View: adview
  Password: False
  Member of host-groups: wsclients
  Indirect Member of HBAC rule: allow_client
  Keytab: True
  Managed by: client.lab.example.net

[student@idm ~]$ ipa hbacrule-show allow_all
  Rule name: allow_all
  User category: all
  Host category: all
  Service category: all
  Description: Allow all users to access any host from any host
  Enabled: FALSE

[student@idm ~]$ logout
(ansible2.9)[student@workstation ~]$
```

This concludes the section.

Guided Exercise: Manage SUDO rules in IdM

Chapter 18. Outcomes

You should be able to:

- Create, delete and SUDO rules
- Associate SUDO rules with host or hostgroups

18.1. Prerequisites

You have successfully completed the previous Guided Exercises.

Opening two tabs in the Gnome Terminal can help you to easily switch between the `idm-inventory` and `idm-code` directories.

Chapter 19. Instructions

1) Login to your workstation as **student** user and open a new terminal. Ensure that you use the `ansible2.9` Python virtual environment.

```
[student@workstation ~]$ source ~/venvs/ansible2.9/bin/activate
(ansible2.9)[student@workstation ~]$
```

2) Navigate to the `workshop/idm-code` directory.

```
(ansible2.9)[student@workstation ~]$ cd ~/workshop/idm-code
```

3) Examing the role `rhidm_manage_sudo`

3.1) Examine the default values of the role's variables. Note the name and the structure of the variables used by the role.

```
(ansible2.9)[student@workstation ~]$ cat ~/workshop/idm-
code/roles/rhidm_manage_sudo/defaults/main.yml
```

3.2 Examine the task file to identify the mandatory variables

```
(ansible2.9)[student@workstation ~]$ cat ~/workshop/idm-
code/roles/rhidm_manage_sudo/tasks/main.yml
```

5) Create a list of dictionaries that will represent the SUDO rules as defined in the following tables:

SUDO Group Name	Description	Commands
Systemctl NetworkManager	All systemctl related commands for NetworkManager	• /usr/bin/systemctl start NetworkManager • /usr/bin/systemctl status NetworkManager • /usr/bin/systemctl status -l NetworkManager • /usr/bin/systemctl stop NetworkManager • /usr/bin/systemctl restart NetworkManager • /usr/bin/systemctl reload NetworkManager

SUDO Rule Name	Description	User Groups	Host Groups
sr_sudo_nm	Allow sudo to NetworkManager	wsgroup01	wsclients

Users should not be asked for their password.

5.1 Create a new file in the IdM inventory that will store the sudo information.

```
(ansible2.9)[student@workstation ~]$ vim ~/workshop/idm-
inventory/group_vars/rhidms/rhidm_sudo_data.yml
```

```
---
rhidm_sudo_cmd_groups:
  - sudo_cmdg_name: Systemctl NetworkManager
    description: "All systemctl related commands for NetworkManager"
    cmds:
      - sudocmd: /usr/bin/systemctl start NetworkManager
        description: Start NetworkManager
      - sudocmd: /usr/bin/systemctl status NetworkManager
        description: Status of NetworkManager
      - sudocmd: /usr/bin/systemctl status -l NetworkManager
        description: NetworkManager Long lines Status
      - sudocmd: /usr/bin/systemctl stop NetworkManager
        description: Stop NetworkManager
      - sudocmd: /usr/bin/systemctl restart NetworkManager
        description: Restart NetworkManager
      - sudocmd: /usr/bin/systemctl reload NetworkManager
        description: Reload NetworkManager

rhidm_sudo_rules:
  - name: "sr_sudo_nm"
    description: "Allow sudo to NetworkManager"
    state: present
    usergroup:
      - wsgroup01
    hostgroup:
      - wsclients
    cmdgroup:
      - "Systemctl NetworkManager"
    sudoopt:
      - '!authenticate'
...
```

7) Include the role `rhidm_manage_sudo` in your playbook and execute it

7.1) Modify the playbook `type__rhidm__configure.yml` to include the role `rhidm_manage_users`

```
[student@workstation ~]$ vim  /home/student/workshop/idm-code/type__rhidm
__configure.yml
---
- name: Configure the RH IDM domain
  hosts: "{{ __config_host | groups['rhidms_master'][0] | default('localhost') }}"
  roles:
    - role: rhidm_configure
      tags: idm_conf

    - role: rhidm_manage_users
      tags: idm_users

    - role: rhidm_manage_groups
      tags: idm_groups

    - role: rhidm_manage_hosts
      tags: idm_hosts

    - role: rhidm_manage_hbac
      tags: idm_hbac

    - role: rhidm_manage_sudo
      tags: idm_sudo
...
```

7.2) Execute the playbook to manage hosts

```
(ansible2.9)[student@workstation ~]$ cd ~/workshop/idm-code/
(ansible2.9)[student@workstation ~]$ ansible-playbook --ask-vault-pass \
 -i ../idm-inventory ./type__rhidm__configure.yml

...output ommitted...

TASK [rhidm_manage_sudo : Manage SUDO Commands]
**********************************************************************************
changed: [idm.lab.example.net] => (item=/usr/bin/systemctl start NetworkManager)
changed: [idm.lab.example.net] => (item=/usr/bin/systemctl status NetworkManager)
changed: [idm.lab.example.net] => (item=/usr/bin/systemctl status -l NetworkManager)
changed: [idm.lab.example.net] => (item=/usr/bin/systemctl stop NetworkManager)
changed: [idm.lab.example.net] => (item=/usr/bin/systemctl restart NetworkManager)
changed: [idm.lab.example.net] => (item=/usr/bin/systemctl reload NetworkManager)

TASK [rhidm_manage_sudo : Manage SUDO Command Groups]
****************************************************************************
changed: [idm.lab.example.net] => (item=Systemctl NetworkManager)

TASK [rhidm_manage_sudo : Manage sudo RULES]
*************************************************************************************
changed: [idm.lab.example.net] => (item=sr_sudo_nm)
```

```
PLAY RECAP
*************************************************************************************
*********************************
idm.lab.example.net        : ok=20   changed:=3    unreachable=0    failed=0
skipped=3    rescued=0    ignored=0
```

8) Login to idm client with the user `wshop01` and execute the commands:

```
[student@workstation ~]$ ssh wshop01@client
Last login: Sun Mar  5 23:38:06 2023 from 172.25.250.254
-sh-4.2$ sudo -l
Matching Defaults entries for wshop01@lab.example.net on client:
    !visiblepw, always_set_home, match_group_by_gid, env_reset, env_keep="COLORS
DISPLAY HOSTNAME HISTSIZE KDEDIR LS_COLORS", env_keep+="MAIL
    PS1 PS2 QTDIR USERNAME LANG LC_ADDRESS LC_CTYPE", env_keep+="LC_COLLATE
LC_IDENTIFICATION LC_MEASUREMENT LC_MESSAGES",
    env_keep+="LC_MONETARY LC_NAME LC_NUMERIC LC_PAPER LC_TELEPHONE",
env_keep+="LC_TIME LC_ALL LANGUAGE LINGUAS _XKB_CHARSET XAUTHORITY",
    secure_path=/sbin\:/bin\:/usr/sbin\:/usr/bin\:/usr/local/sbin\:/usr/local/bin

User wshop01@lab.example.net may run the following commands on client:
    (ALL : ALL) NOPASSWD: /usr/bin/systemctl stop NetworkManager, /usr/bin/systemctl
start NetworkManager, /usr/bin/systemctl status
        NetworkManager, /usr/bin/systemctl status -l NetworkManager,
/usr/bin/systemctl restart NetworkManager, /usr/bin/systemctl reload
        NetworkManager

-sh-4.2$ logout
(ansible2.9)[student@workstation ~]$
```

This concludes the section.