



Red Hat Enterprise Linux 8

Using Ansible to install and manage Identity Management

Installing, configuring, managing, and maintaining Identity Management in Red Hat Enterprise Linux 8 using Red Hat Ansible Engine

Red Hat Enterprise Linux 8 Using Ansible to install and manage Identity Management

Installing, configuring, managing, and maintaining Identity Management in Red Hat Enterprise Linux 8 using Red Hat Ansible Engine

Legal Notice

Copyright © 2021 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This documentation collection provides instructions on how to effectively use Ansible playbooks to install, configure, manage and maintain Identity Management on Red Hat Enterprise Linux 8.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE	8
PROVIDING FEEDBACK ON RED HAT DOCUMENTATION	9
CHAPTER 1. ANSIBLE TERMINOLOGY	10
CHAPTER 2. INSTALLING AN IDENTITY MANAGEMENT SERVER USING AN ANSIBLE PLAYBOOK	11
2.1. ANSIBLE AND ITS ADVANTAGES FOR INSTALLING IDM	11
Advantages of using Ansible to install IdM	11
2.2. IDM SERVER INSTALLATION USING AN ANSIBLE PLAYBOOK	11
Overview	11
2.3. INSTALLING THE ANSIBLE-FREEIPA PACKAGE	12
2.4. ANSIBLE ROLES LOCATION IN THE FILE SYSTEM	12
2.5. DEPLOYING AN IDM SERVER WITH AN INTEGRATED CA AS THE ROOT CA USING AN ANSIBLE PLAYBOOK	13
2.5.1. Setting the parameters for a deployment with an integrated CA as the root CA	13
2.5.2. Deploying an IdM server with an integrated CA as the root CA using an Ansible playbook	15
2.6. DEPLOYING AN IDM SERVER WITH AN EXTERNAL CA AS THE ROOT CA USING AN ANSIBLE PLAYBOOK	15
2.6.1. Setting the parameters for a deployment with an external CA as the root CA	15
2.6.2. Deploying an IdM server with an external CA as the root CA using an Ansible playbook	18
CHAPTER 3. INSTALLING AN IDENTITY MANAGEMENT REPLICA USING AN ANSIBLE PLAYBOOK	19
3.1. IDM REPLICA INSTALLATION USING AN ANSIBLE PLAYBOOK	19
Overview	19
3.2. SETTING THE PARAMETERS OF THE IDM REPLICA DEPLOYMENT	19
3.2.1. Specifying the base, server and client variables for installing the IdM replica	20
3.2.2. Specifying the credentials for installing the IdM replica using an Ansible playbook	22
3.3. DEPLOYING AN IDM REPLICA USING AN ANSIBLE PLAYBOOK	23
CHAPTER 4. INSTALLING AN IDENTITY MANAGEMENT CLIENT USING AN ANSIBLE PLAYBOOK	25
4.1. IDM CLIENT INSTALLATION USING AN ANSIBLE PLAYBOOK	25
Overview	25
4.2. SETTING THE PARAMETERS OF THE IDM CLIENT DEPLOYMENT	25
4.2.1. Setting the parameters of the inventory file for the autodiscovery client installation mode	26
4.2.2. Setting the parameters of the inventory file when autodiscovery is not possible during client installation	28
4.2.3. Checking the parameters in the install-client.yml file	30
4.2.4. Authorization options for IdM client enrollment using an Ansible playbook	30
4.3. DEPLOYING AN IDM CLIENT USING AN ANSIBLE PLAYBOOK	32
4.4. TESTING AN IDENTITY MANAGEMENT CLIENT AFTER ANSIBLE INSTALLATION	33
4.5. UNINSTALLING AN IDM CLIENT USING AN ANSIBLE PLAYBOOK	33
CHAPTER 5. PREPARING YOUR ENVIRONMENT FOR MANAGING IDM USING ANSIBLE PLAYBOOKS	35
CHAPTER 6. CONFIGURING GLOBAL IDM SETTINGS USING ANSIBLE PLAYBOOKS	37
6.1. RETRIEVING IDM CONFIGURATION USING AN ANSIBLE PLAYBOOK	37
6.2. CONFIGURING THE IDM CA RENEWAL SERVER USING AN ANSIBLE PLAYBOOK	39
6.3. CONFIGURING THE DEFAULT SHELL FOR IDM USERS USING AN ANSIBLE PLAYBOOK	40
CHAPTER 7. MANAGING USER ACCOUNTS USING ANSIBLE PLAYBOOKS	42
7.1. USER LIFE CYCLE	42
7.2. ENSURING THE PRESENCE OF AN IDM USER USING AN ANSIBLE PLAYBOOK	43
7.3. ENSURING THE PRESENCE OF MULTIPLE IDM USERS USING ANSIBLE PLAYBOOKS	45

7.4. ENSURING THE PRESENCE OF MULTIPLE IDM USERS FROM A JSON FILE USING ANSIBLE PLAYBOOKS	46
7.5. ENSURING THE ABSENCE OF USERS USING ANSIBLE PLAYBOOKS	48
CHAPTER 8. MANAGING USER GROUPS USING ANSIBLE PLAYBOOKS	50
8.1. THE DIFFERENT GROUP TYPES IN IDM	50
8.2. DIRECT AND INDIRECT GROUP MEMBERS	51
8.3. ENSURING THE PRESENCE OF IDM GROUPS AND GROUP MEMBERS USING ANSIBLE PLAYBOOKS	52
8.4. ENSURING THE PRESENCE OF MEMBER MANAGERS IN IDM USER GROUPS USING ANSIBLE PLAYBOOKS	53
8.5. ENSURING THE ABSENCE OF MEMBER MANAGERS IN IDM USER GROUPS USING ANSIBLE PLAYBOOKS	54
CHAPTER 9. USING ANSIBLE PLAYBOOKS TO MANAGE SELF-SERVICE RULES IN IDM	57
9.1. SELF-SERVICE ACCESS CONTROL IN IDM	57
9.2. USING ANSIBLE TO ENSURE THAT A SELF-SERVICE RULE IS PRESENT	57
9.3. USING ANSIBLE TO ENSURE THAT A SELF-SERVICE RULE IS ABSENT	59
9.4. USING ANSIBLE TO ENSURE THAT A SELF-SERVICE RULE HAS SPECIFIC ATTRIBUTES	60
9.5. USING ANSIBLE TO ENSURE THAT A SELF-SERVICE RULE DOES NOT HAVE SPECIFIC ATTRIBUTES	62
CHAPTER 10. DELEGATING PERMISSIONS TO USER GROUPS TO MANAGE USERS USING ANSIBLE PLAYBOOKS	64
10.1. DELEGATION RULES	64
10.2. CREATING AN ANSIBLE INVENTORY FILE FOR IDM	64
10.3. USING ANSIBLE TO ENSURE THAT A DELEGATION RULE IS PRESENT	65
10.4. USING ANSIBLE TO ENSURE THAT A DELEGATION RULE IS ABSENT	67
10.5. USING ANSIBLE TO ENSURE THAT A DELEGATION RULE HAS SPECIFIC ATTRIBUTES	68
10.6. USING ANSIBLE TO ENSURE THAT A DELEGATION RULE DOES NOT HAVE SPECIFIC ATTRIBUTES	70
CHAPTER 11. USING ANSIBLE PLAYBOOKS TO MANAGE ROLE-BASED ACCESS CONTROL IN IDM	72
11.1. PERMISSIONS IN IDM	72
11.2. DEFAULT MANAGED PERMISSIONS	73
11.3. PRIVILEGES IN IDM	75
11.4. ROLES IN IDM	75
11.5. PREDEFINED ROLES IN IDENTITY MANAGEMENT	75
11.6. USING ANSIBLE TO ENSURE AN IDM RBAC ROLE WITH PRIVILEGES IS PRESENT	76
11.7. USING ANSIBLE TO ENSURE AN IDM RBAC ROLE IS ABSENT	78
11.8. USING ANSIBLE TO ENSURE THAT A GROUP OF USERS IS ASSIGNED TO AN IDM RBAC ROLE	79
11.9. USING ANSIBLE TO ENSURE THAT SPECIFIC USERS ARE NOT ASSIGNED TO AN IDM RBAC ROLE	81
11.10. USING ANSIBLE TO ENSURE A SERVICE IS A MEMBER OF AN IDM RBAC ROLE	82
11.11. USING ANSIBLE TO ENSURE A HOST IS A MEMBER OF AN IDM RBAC ROLE	84
11.12. USING ANSIBLE TO ENSURE A HOST GROUP IS A MEMBER OF AN IDM RBAC ROLE	85
CHAPTER 12. USING ANSIBLE PLAYBOOKS TO MANAGE RBAC PRIVILEGES	88
12.1. USING ANSIBLE TO ENSURE A CUSTOM IDM RBAC PRIVILEGE IS PRESENT	88
12.2. USING ANSIBLE TO ENSURE MEMBER PERMISSIONS ARE PRESENT IN A CUSTOM IDM RBAC PRIVILEGE	89
12.3. USING ANSIBLE TO ENSURE AN IDM RBAC PRIVILEGE DOES NOT INCLUDE A PERMISSION	91
12.4. USING ANSIBLE TO RENAME A CUSTOM IDM RBAC PRIVILEGE	92
12.5. USING ANSIBLE TO ENSURE AN IDM RBAC PRIVILEGE IS ABSENT	94
CHAPTER 13. USING ANSIBLE PLAYBOOKS TO MANAGE RBAC PERMISSIONS IN IDM	96
13.1. USING ANSIBLE TO ENSURE AN RBAC PERMISSION IS PRESENT	96

13.2. USING ANSIBLE TO ENSURE AN RBAC PERMISSION WITH AN ATTRIBUTE IS PRESENT	98
13.3. USING ANSIBLE TO ENSURE AN RBAC PERMISSION IS ABSENT	99
13.4. USING ANSIBLE TO ENSURE AN ATTRIBUTE IS A MEMBER OF AN IDM RBAC PERMISSION	101
13.5. USING ANSIBLE TO ENSURE AN ATTRIBUTE IS NOT A MEMBER OF AN IDM RBAC PERMISSION	102
13.6. USING ANSIBLE TO RENAME AN IDM RBAC PERMISSION	103
13.7. ADDITIONAL RESOURCES	105
CHAPTER 14. USING ANSIBLE TO MANAGE THE REPLICATION TOPOLOGY IN IDM	106
14.1. USING ANSIBLE TO ENSURE A REPLICATION AGREEMENT EXISTS IN IDM	106
14.2. USING ANSIBLE TO ENSURE REPLICATION AGREEMENTS EXIST BETWEEN MULTIPLE IDM REPLICAS	108
14.3. USING ANSIBLE TO CHECK IF A REPLICATION AGREEMENT EXISTS BETWEEN TWO REPLICAS	110
14.4. USING ANSIBLE TO VERIFY THAT A TOPOLOGY SUFFIX EXISTS IN IDM	112
14.5. USING ANSIBLE TO REINITIALIZE AN IDM REPLICA	113
14.6. USING ANSIBLE TO ENSURE A REPLICATION AGREEMENT IS ABSENT IN IDM	115
14.7. ADDITIONAL RESOURCES	116
CHAPTER 15. MANAGING HOSTS USING ANSIBLE PLAYBOOKS	117
15.1. HOSTS IN IDM	117
15.2. HOST ENROLLMENT	118
15.2.1. User privileges required for host enrollment	118
User privileges for optionally manually creating a host entry in IdM LDAP	118
User privileges for joining the client to the IdM domain	118
15.2.2. Enrollment and authentication of IdM hosts and users: comparison	119
15.2.3. Alternative authentication options for IdM hosts	120
15.3. ENSURING THE PRESENCE OF AN IDM HOST ENTRY WITH FQDN USING ANSIBLE PLAYBOOKS	120
15.4. ENSURING THE PRESENCE OF AN IDM HOST ENTRY WITH DNS INFORMATION USING ANSIBLE PLAYBOOKS	122
15.5. ENSURING THE PRESENCE OF MULTIPLE IDM HOST ENTRIES WITH RANDOM PASSWORDS USING ANSIBLE PLAYBOOKS	124
15.6. ENSURING THE PRESENCE OF AN IDM HOST ENTRY WITH MULTIPLE IP ADDRESSES USING ANSIBLE PLAYBOOKS	125
15.7. ENSURING THE ABSENCE OF AN IDM HOST ENTRY USING ANSIBLE PLAYBOOKS	127
CHAPTER 16. MANAGING HOST GROUPS USING ANSIBLE PLAYBOOKS	129
16.1. HOST GROUPS IN IDM	129
16.2. ENSURING THE PRESENCE OF IDM HOST GROUPS USING ANSIBLE PLAYBOOKS	129
16.3. ENSURING THE PRESENCE OF HOSTS IN IDM HOST GROUPS USING ANSIBLE PLAYBOOKS	131
16.4. NESTING IDM HOST GROUPS USING ANSIBLE PLAYBOOKS	132
16.5. ENSURING THE PRESENCE OF MEMBER MANAGERS IN IDM HOST GROUPS USING ANSIBLE PLAYBOOKS	134
16.6. ENSURING THE ABSENCE OF HOSTS FROM IDM HOST GROUPS USING ANSIBLE PLAYBOOKS	135
16.7. ENSURING THE ABSENCE OF NESTED HOST GROUPS FROM IDM HOST GROUPS USING ANSIBLE PLAYBOOKS	137
16.8. ENSURING THE ABSENCE OF IDM HOST GROUPS USING ANSIBLE PLAYBOOKS	138
16.9. ENSURING THE ABSENCE OF MEMBER MANAGERS FROM IDM HOST GROUPS USING ANSIBLE PLAYBOOKS	139
CHAPTER 17. DEFINING IDM PASSWORD POLICIES	142
17.1. WHAT IS A PASSWORD POLICY	142
17.2. PASSWORD POLICIES IN IDM	142
17.3. ENSURING THE PRESENCE OF A PASSWORD POLICY IN IDM USING AN ANSIBLE PLAYBOOK	144
17.4. ADDITIONAL PASSWORD POLICY OPTIONS IN IDM	145
17.5. APPLYING ADDITIONAL PASSWORD POLICY OPTIONS TO AN IDM GROUP	146

CHAPTER 18. GRANTING SUDO ACCESS TO AN IDM USER ON AN IDM CLIENT	149
18.1. SUDO ACCESS ON AN IDM CLIENT	149
18.2. GRANTING SUDO ACCESS TO AN IDM USER ON AN IDM CLIENT USING THE CLI	149
18.3. GRANTING SUDO ACCESS TO AN IDM USER ON AN IDM CLIENT USING THE IDM WEB UI	151
18.4. CREATING A SUDO RULE ON THE CLI THAT RUNS A COMMAND AS A SERVICE ACCOUNT ON AN IDM CLIENT	154
18.5. CREATING A SUDO RULE IN THE IDM WEBUI THAT RUNS A COMMAND AS A SERVICE ACCOUNT ON AN IDM CLIENT	156
18.6. ENABLING GSSAPI AUTHENTICATION FOR SUDO ON AN IDM CLIENT	162
18.7. ENABLING GSSAPI AUTHENTICATION AND ENFORCING KERBEROS AUTHENTICATION INDICATORS FOR SUDO ON AN IDM CLIENT	164
18.8. SSSD OPTIONS CONTROLLING GSSAPI AUTHENTICATION FOR PAM SERVICES	166
18.9. TROUBLESHOOTING GSSAPI AUTHENTICATION FOR SUDO	167
18.10. USING AN ANSIBLE PLAYBOOK TO ENSURE SUDO ACCESS FOR AN IDM USER ON AN IDM CLIENT	169
CHAPTER 19. ENSURING THE PRESENCE OF HOST-BASED ACCESS CONTROL RULES IN IDM USING ANSIBLE PLAYBOOKS	172
19.1. HOST-BASED ACCESS CONTROL RULES IN IDM	172
19.2. ENSURING THE PRESENCE OF AN HBAC RULE IN IDM USING AN ANSIBLE PLAYBOOK	172
CHAPTER 20. VAULTS IN IDM	174
20.1. VAULTS AND THEIR BENEFITS	174
20.2. VAULT OWNERS, MEMBERS, AND ADMINISTRATORS	175
20.3. STANDARD, SYMMETRIC, AND ASYMMETRIC VAULTS	176
20.4. USER, SERVICE, AND SHARED VAULTS	176
20.5. VAULT CONTAINERS	176
20.6. BASIC IDM VAULT COMMANDS	177
20.7. INSTALLING THE KEY RECOVERY AUTHORITY IN IDM	177
CHAPTER 21. USING ANSIBLE TO MANAGE IDM USER VAULTS: STORING AND RETRIEVING SECRETS	179
21.1. ENSURING THE PRESENCE OF A STANDARD USER VAULT IN IDM USING ANSIBLE	179
21.2. ARCHIVING A SECRET IN A STANDARD USER VAULT IN IDM USING ANSIBLE	180
21.3. RETRIEVING A SECRET FROM A STANDARD USER VAULT IN IDM USING ANSIBLE	182
CHAPTER 22. USING ANSIBLE TO MANAGE IDM SERVICE VAULTS: STORING AND RETRIEVING SECRETS	184
22.1. ENSURING THE PRESENCE OF AN ASYMMETRIC SERVICE VAULT IN IDM USING ANSIBLE	185
22.2. ADDING MEMBER SERVICES TO AN ASYMMETRIC VAULT USING ANSIBLE	186
22.3. STORING AN IDM SERVICE SECRET IN AN ASYMMETRIC VAULT USING ANSIBLE	188
22.4. RETRIEVING A SERVICE SECRET FOR AN IDM SERVICE USING ANSIBLE	189
22.5. CHANGING AN IDM SERVICE VAULT SECRET WHEN COMPROMISED USING ANSIBLE	192
CHAPTER 23. ENSURING THE PRESENCE AND ABSENCE OF SERVICES IN IDM USING ANSIBLE	196
23.1. ENSURING THE PRESENCE OF AN HTTP SERVICE IN IDM USING AN ANSIBLE PLAYBOOK	196
23.2. ENSURING THE PRESENCE OF AN HTTP SERVICE IN IDM ON A NON-IDM CLIENT USING AN ANSIBLE PLAYBOOK	198
23.3. ENSURING THE PRESENCE OF AN HTTP SERVICE ON AN IDM CLIENT WITHOUT DNS USING AN ANSIBLE PLAYBOOK	199
23.4. ENSURING THE PRESENCE OF AN EXTERNALLY SIGNED CERTIFICATE IN AN IDM SERVICE ENTRY USING AN ANSIBLE PLAYBOOK	201
23.5. USING AN ANSIBLE PLAYBOOK TO ALLOW IDM USERS, GROUPS, HOSTS, OR HOST GROUPS TO CREATE A KEYTAB OF A SERVICE	203
23.6. USING AN ANSIBLE PLAYBOOK TO ALLOW IDM USERS, GROUPS, HOSTS, OR HOST GROUPS TO RETRIEVE A KEYTAB OF A SERVICE	205
23.7. ENSURING THE PRESENCE OF A KERBEROS PRINCIPAL ALIAS OF A SERVICE USING AN ANSIBLE	

PLAYBOOK	207
23.8. ENSURING THE ABSENCE OF AN HTTP SERVICE IN IDM USING AN ANSIBLE PLAYBOOK	209
CHAPTER 24. MANAGING GLOBAL DNS CONFIGURATION IN IDM USING ANSIBLE PLAYBOOKS	211
24.1. HOW IDM ENSURES THAT GLOBAL FORWARDERS FROM /ETC/RESOLV.CONF ARE NOT REMOVED BY NETWORKMANAGER	211
24.2. ENSURING THE PRESENCE OF A DNS GLOBAL FORWARDER IN IDM USING ANSIBLE	212
24.3. ENSURING THE ABSENCE OF A DNS GLOBAL FORWARDER IN IDM USING ANSIBLE	214
24.4. DNS FORWARD POLICIES IN IDM	215
24.5. USING AN ANSIBLE PLAYBOOK TO ENSURE THAT THE FORWARD FIRST POLICY IS SET IN IDM DNS GLOBAL CONFIGURATION	216
24.6. USING AN ANSIBLE PLAYBOOK TO ENSURE THAT GLOBAL FORWARDERS ARE DISABLED IN IDM DNS	217
24.7. USING AN ANSIBLE PLAYBOOK TO ENSURE THAT SYNCHRONIZATION OF FORWARD AND REVERSE LOOKUP ZONES IS DISABLED IN IDM DNS	219
CHAPTER 25. USING ANSIBLE PLAYBOOKS TO MANAGE IDM DNS ZONES	221
25.1. SUPPORTED DNS ZONE TYPES	221
25.2. CONFIGURATION ATTRIBUTES OF PRIMARY IDM DNS ZONES	222
25.3. USING ANSIBLE TO CREATE A PRIMARY ZONE IN IDM DNS	224
25.4. USING AN ANSIBLE PLAYBOOK TO ENSURE THE PRESENCE OF A PRIMARY DNS ZONE IN IDM WITH MULTIPLE VARIABLES	225
25.5. USING AN ANSIBLE PLAYBOOK TO ENSURE THE PRESENCE OF A ZONE FOR REVERSE DNS LOOKUP WHEN AN IP ADDRESS IS GIVEN	228
CHAPTER 26. USING ANSIBLE TO MANAGE DNS LOCATIONS IN IDM	230
26.1. DNS-BASED SERVICE DISCOVERY	230
26.2. DEPLOYMENT CONSIDERATIONS FOR DNS LOCATIONS	231
26.3. DNS TIME TO LIVE (TTL)	231
26.4. USING ANSIBLE TO ENSURE AN IDM LOCATION IS PRESENT	231
26.5. USING ANSIBLE TO ENSURE AN IDM LOCATION IS ABSENT	233
26.6. ADDITIONAL RESOURCES	234
CHAPTER 27. MANAGING DNS FORWARDING IN IDM	235
27.1. THE TWO ROLES OF AN IDM DNS SERVER	235
27.2. DNS FORWARD POLICIES IN IDM	236
27.3. ADDING A GLOBAL FORWARDER IN THE IDM WEB UI	236
27.4. ADDING A GLOBAL FORWARDER IN THE CLI	239
27.5. ADDING A DNS FORWARD ZONE IN THE IDM WEB UI	240
27.6. ADDING A DNS FORWARD ZONE IN THE CLI	243
27.7. ESTABLISHING A DNS GLOBAL FORWARDER IN IDM USING ANSIBLE	244
27.8. ENSURING THE PRESENCE OF A DNS GLOBAL FORWARDER IN IDM USING ANSIBLE	245
27.9. ENSURING THE ABSENCE OF A DNS GLOBAL FORWARDER IN IDM USING ANSIBLE	247
27.10. ENSURING DNS GLOBAL FORWARDERS ARE DISABLED IN IDM USING ANSIBLE	248
27.11. ENSURING THE PRESENCE OF A DNS FORWARD ZONE IN IDM USING ANSIBLE	249
27.12. ENSURING A DNS FORWARD ZONE HAS MULTIPLE FORWARDERS IN IDM USING ANSIBLE	251
27.13. ENSURING A DNS FORWARD ZONE IS DISABLED IN IDM USING ANSIBLE	253
27.14. ENSURING THE ABSENCE OF A DNS FORWARD ZONE IN IDM USING ANSIBLE	254
CHAPTER 28. USING ANSIBLE TO MANAGE DNS RECORDS IN IDM	257
28.1. DNS RECORDS IN IDM	257
28.2. COMMON IPA DNSRECORD-* OPTIONS	258
28.3. ENSURING THE PRESENCE OF A AND AAAA DNS RECORDS IN IDM USING ANSIBLE	260
28.4. ENSURING THE PRESENCE OF A AND PTR DNS RECORDS IN IDM USING ANSIBLE	262
28.5. ENSURING THE PRESENCE OF MULTIPLE DNS RECORDS IN IDM USING ANSIBLE	264

28.6. ENSURING THE PRESENCE OF MULTIPLE CNAME RECORDS IN IDM USING ANSIBLE	265
28.7. ENSURING THE PRESENCE OF AN SRV RECORD IN IDM USING ANSIBLE	267

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see [our CTO Chris Wright's message](#).

In Identity Management, planned terminology replacements include:

- ***block list*** replaces *blacklist*
- ***allow list*** replaces *whitelist*
- ***secondary*** replaces *slave*
- The word *master* is being replaced with more precise language, depending on the context:
 - ***IdM server*** replaces *IdM master*
 - ***CA renewal server*** replaces *CA renewal master*
 - ***CRL publisher server*** replaces *CRL master*
 - ***multi-supplier*** replaces *multi-master*

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Please let us know how we could make it better. To do so:

- For simple comments on specific passages:
 1. Make sure you are viewing the documentation in the *Multi-page HTML* format. In addition, ensure you see the **Feedback** button in the upper right corner of the document.
 2. Use your mouse cursor to highlight the part of text that you want to comment on.
 3. Click the **Add Feedback** pop-up that appears below the highlighted text.
 4. Follow the displayed instructions.
- For submitting more complex feedback, create a Bugzilla ticket:
 1. Go to the [Bugzilla](#) website.
 2. As the Component, use **Documentation**.
 3. Fill in the **Description** field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.
 4. Click **Submit Bug**.

CHAPTER 1. ANSIBLE TERMINOLOGY

The chapters in this title use the official Ansible terminology. If you are not familiar with the terminology, read the [official Ansible upstream documentation](#) before proceeding, especially the following sections:

- The [Basic concepts in Ansible](#) section provides an overview of the most commonly used concepts in Ansible.
- The [User guide](#) outlines the most common situations and questions when starting to use Ansible, such as using the command line; working with an inventory; interacting with data; writing tasks, plays, and playbooks; and executing playbooks.
- [How to build your inventory](#) offers tips on how to design your inventory. An inventory is a list or a group of lists that Ansible uses to work against multiple managed nodes or hosts in your infrastructure.
- [Intro to playbooks](#) introduces the concept of an Ansible playbook as a repeatable and re-usable system for managing configurations, deploying machines, and deploying complex applications.
- The [Ansible roles](#) section explains how to automate loading variables, tasks, and handlers based on a known file structure.
- The [Glossary](#) explains terms that are used elsewhere in the Ansible documentation.

CHAPTER 2. INSTALLING AN IDENTITY MANAGEMENT SERVER USING AN ANSIBLE PLAYBOOK

2.1. ANSIBLE AND ITS ADVANTAGES FOR INSTALLING IDM

Ansible is an automation tool used to configure systems, deploy software, and perform rolling updates. Ansible includes support for Identity Management (IdM), and you can use Ansible modules to automate installation tasks such as the setup of an IdM server, replica, client, or an entire IdM topology.

Advantages of using Ansible to install IdM

The following list presents advantages of installing Identity Management using Ansible in contrast to manual installation.

- You do not need to log into the managed node.
- You do not need to configure settings on each host to be deployed individually. Instead, you can have one inventory file to deploy a complete cluster.
- You can reuse an inventory file later for management tasks, for example to add users and hosts. You can reuse an inventory file even for such tasks as are not related to IdM.

2.2. IDM SERVER INSTALLATION USING AN ANSIBLE PLAYBOOK

The following sections describe how to configure a system as an IdM server by using [Ansible](#). Configuring a system as an IdM server establishes an IdM domain and enables the system to offer IdM services to IdM clients. The deployment is managed by the **ipaserver** Ansible role.



NOTE

Before installing an IdM server using Ansible, ensure that you understand [Ansible](#) and IdM concepts. Ensure that you understand the following terms that are used in this chapter:

- Ansible roles
- Ansible nodes
- Ansible inventory
- Ansible tasks
- Ansible modules
- Ansible plays and playbooks

Overview

The installation consists of the following parts:

1. [Installing the ansible-freeipa package](#)
2. [Deploying an IdM server with an integrated CA using an Ansible playbook](#)
3. [Deploying an IdM server with an external CA using an Ansible-playbook](#)

2.3. INSTALLING THE ANSIBLE-FREEIPA PACKAGE

Prerequisites

On the **managed node**:

- Ensure that the managed node is a Red Hat Enterprise Linux 8 system with a static IP address and a working package manager.

On the **controller**:

- Ensure that the controller is a Red Hat Enterprise Linux system with a valid subscription. If this is not the case, see the official Ansible documentation [Installation guide](#) for alternative installation instructions.
- Ensure that you can reach the managed node over the **SSH** protocol from the controller. Check that the managed node is listed in the **/root/.ssh/known_hosts** file of the controller.

Procedure

Run the following procedure on the Ansible controller.

1. Enable the required repository:

```
# subscription-manager repos --enable ansible-2.8-for-rhel-8-x86_64-rpms
```

2. Install Ansible:

```
# yum install ansible
```

3. Install the IdM Ansible roles:

```
# yum install ansible-freeipa
```

The roles are installed to the **/usr/share/ansible/roles/** directory.

2.4. ANSIBLE ROLES LOCATION IN THE FILE SYSTEM

By default the **ansible-freeipa** roles are installed to the **/usr/share/ansible/roles/** directory. The structure of the **ansible-freeipa** package is as follows:

- The **/usr/share/ansible/roles/** directory stores the **ipaserver**, **ipareplica**, and **ipacient** roles on the Ansible controller. Each role directory stores examples, a basic overview, the licence and documentation about the role in a README.md Markdown file.

```
[root@server]# ls -l /usr/share/ansible/roles/
ipaclient
ipareplica
ipaserver
```

- The **/usr/share/doc/ansible-freeipa/** directory stores the documentation about individual roles and the topology in README.md Markdown files. It also stores the **playbooks/** subdirectory (see below).

```
[root@server]# ls -l /usr/share/doc/ansible-freeipa/
```



```
playbooks
README-client.md
README.md
README-replica.md
README-server.md
README-topology.md
```

- The `/usr/share/doc/ansible-freeipa/playbooks/` directory stores the example playbooks:

```
[root@server]# ls -l /usr/share/doc/ansible-freeipa/playbooks/
install-client.yml
install-cluster.yml
install-replica.yml
install-server.yml
uninstall-client.yml
uninstall-cluster.yml
uninstall-replica.yml
uninstall-server.yml
```

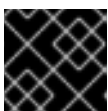
2.5. DEPLOYING AN IDM SERVER WITH AN INTEGRATED CA AS THE ROOT CA USING AN ANSIBLE PLAYBOOK

2.5.1. Setting the parameters for a deployment with an integrated CA as the root CA

Complete this procedure to configure the inventory file for installing an IdM server with an integrated CA as the root CA.

Procedure

1. Open the inventory file for editing. Specify the fully-qualified domain names (**FQDN**) of the host you want to use as an IdM server. Ensure that the **FQDN** meets the following criteria:
 - Only alphanumeric characters and hyphens (-) are allowed. For example, underscores are not allowed and can cause DNS failures.
 - The host name must be all lower-case.
2. Specify the IdM domain and realm information.
3. Specify if you want the IdM server to have an integrated DNS and if you want it to use forwarders from the `/etc/resolv.conf` file.
4. Specify the passwords for **admin** and for the **Directory Manager**. Use the Ansible Vault to store the password, and reference the Vault file from the playbook file. Alternatively and less securely, specify the passwords directly in the inventory file.
5. (Optional) Specify a custom **firewalld** zone to be used by the IdM server. If you do not set a custom zone, IdM will add its services to the default **firewalld** zone. The predefined default zone is **public**.



IMPORTANT

The specified **firewalld** zone must exist and be permanent.

Example of an inventory file with the required server information (excluding the passwords)

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

[ipaserver:vars]
ipaserver_domain=idm.example.com
ipaserver_realm=IDM.EXAMPLE.COM
ipaserver_setup_dns=yes
ipaserver_auto_forwarders=yes
[...]
```

Example of an inventory file with the required server information (including the passwords)

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

[ipaserver:vars]
ipaserver_domain=idm.example.com
ipaserver_realm=IDM.EXAMPLE.COM
ipaserver_setup_dns=yes
ipaserver_auto_forwarders=yes
ipaadmin_password=MySecretPassword123
ipadm_password=MySecretPassword234

[...]
```

Example of an inventory file with a custom `firewalld` zone

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

[ipaserver:vars]
ipaserver_domain=idm.example.com
ipaserver_realm=IDM.EXAMPLE.COM
ipaserver_setup_dns=yes
ipaserver_auto_forwarders=yes
ipaadmin_password=MySecretPassword123
ipadm_password=MySecretPassword234
ipaserver_firewalld_zone=custom zone
```

Example playbook to set up an IdM server using admin and Directory Manager passwords stored in an Ansible Vault file

```
---
- name: Playbook to configure IPA server
  hosts: ipaserver
  become: true
  vars_files:
    - playbook_sensitive_data.yml
```

```
roles:
- role: ipaserver
  state: present
```

Example playbook to set up an IdM server using admin and Directory Manager passwords from an inventory file

```
---
- name: Playbook to configure IPA server
  hosts: ipaserver
  become: true

  roles:
  - role: ipaserver
    state: present
```

For details on installing the IdM server and the available options, see https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/installing_identity_management/index#installing-idm.

2.5.2. Deploying an IdM server with an integrated CA as the root CA using an Ansible playbook

Complete this procedure to deploy an IdM server with an integrated certificate authority (CA) as the root CA using an Ansible playbook.

Procedure

- Run the **ansible-playbook** command with the name of the playbook file, for example **install-server.yml**. Specify the inventory file with the **-i** option:

```
$ ansible-playbook -v -i <path_to_inventory_directory>/hosts
<path_to_playbooks_directory>/install-server.yml
```

Specify the level of verbosity by using the **-v**, **-vv**, or **-vvv** option.

You can view the output of the Ansible playbook script on the command-line interface (CLI). The following output shows that the script has run successfully as 0 tasks have failed:

```
PLAY RECAP
server.idm.example.com : ok=18  changed=10  unreachable=0  failed=0  skipped=21
rescued=0  ignored=0
```

You have installed an IdM server on your host using an Ansible playbook.

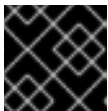
2.6. DEPLOYING AN IDM SERVER WITH AN EXTERNAL CA AS THE ROOT CA USING AN ANSIBLE PLAYBOOK

2.6.1. Setting the parameters for a deployment with an external CA as the root CA

Complete this procedure to configure the inventory file for installing an IdM server with an external CA as the root CA.

Procedure

1. Open the inventory file for editing. Specify the fully-qualified domain names (**FQDN**) of the host you want to use as an IdM server. Ensure that the **FQDN** meets the following criteria:
 - Only alphanumeric characters and hyphens (-) are allowed. For example, underscores are not allowed and can cause DNS failures.
 - The host name must be all lower-case.
2. Specify the IdM domain and realm information.
3. Specify if you want the IdM server to have an integrated DNS and if you want it to use forwarders from the `/etc/resolv.conf` file.
4. Specify the passwords for **admin** and for the **Directory Manager**. Use the Ansible Vault to store the password, and reference the Vault file from the playbook file. Alternatively and less securely, specify the passwords directly in the inventory file.
5. (Optional) Specify a custom **firewalld** zone to be used by the IdM server. If you do not set a custom zone, IdM will add its services to the default **firewalld** zone. The predefined default zone is **public**.



IMPORTANT

The specified **firewalld** zone must exist and be permanent.

Example of an inventory file with the required server information (excluding the passwords)

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

[ipaserver:vars]
ipaserver_domain=idm.example.com
ipaserver_realm=IDM.EXAMPLE.COM
ipaserver_setup_dns=yes
ipaserver_auto_forwarders=yes
[...]
```

Example of an inventory file with the required server information (including the passwords)

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

[ipaserver:vars]
ipaserver_domain=idm.example.com
ipaserver_realm=IDM.EXAMPLE.COM
ipaserver_setup_dns=yes
ipaserver_auto_forwarders=yes
ipaadmin_password=MySecretPassword123
ipadm_password=MySecretPassword234

[...]
```

Example of an inventory file with a custom `firewalld` zone

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

[ipaserver:vars]
ipaserver_domain=idm.example.com
ipaserver_realm=IDM.EXAMPLE.COM
ipaserver_setup_dns=yes
ipaserver_auto_forwarders=yes
ipaadmin_password=MySecretPassword123
ipadm_password=MySecretPassword234
ipaserver_firewalld_zone=custom zone

[...]
```

6. Create a playbook for the first step of the installation. Enter instructions for generating the certificate signing request (CSR) and copying it from the controller to the managed node.

```
---
- name: Playbook to configure IPA server Step 1
  hosts: ipaserver
  become: true
  vars_files:
    - playbook_sensitive_data.yml
  vars:
    ipaserver_external_ca: yes

  roles:
    - role: ipaserver
      state: present

  post_tasks:
    - name: Copy CSR /root/ipa.csr from node to "{{ groups.ipaserver[0] + '-ipa.csr' }}"
      fetch:
        src: /root/ipa.csr
        dest: "{{ groups.ipaserver[0] + '-ipa.csr' }}"
        flat: yes
```

7. Create another playbook for the final step of the installation.

```
---
- name: Playbook to configure IPA server Step -1
  hosts: ipaserver
  become: true
  vars_files:
    - playbook_sensitive_data.yml
  vars:
    ipaserver_external_cert_files: "/root/chain.crt"

  pre_tasks:
    - name: Copy "{{ groups.ipaserver[0] + '-chain.crt' }}" to /root/chain.crt on node
      copy:
        src: "{{ groups.ipaserver[0] + '-chain.crt' }}"
        dest: "/root/chain.crt"
```

```

    force: yes

roles:
- role: ipaserver
  state: present

```

For details on the options available to you when installing an IdM server with an externally signed CA, see https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/installing_identity_management/index#installing-an-ipa-server-with-external-ca_installing-identity-management.

2.6.2. Deploying an IdM server with an external CA as the root CA using an Ansible playbook

Complete this procedure to deploy an IdM server with an external certificate authority (CA) as the root CA using an Ansible playbook.

Procedure

1. Run the **ansible-playbook** command with the name of the playbook file that contains instructions for the first step of the installation, for example **install-server-step1.yml**. Specify the inventory file with the **-i** option:

```
$ ansible-playbook -v -i <path_to_inventory_directory>/host.server
<path_to_playbooks_directory>/install-server-step1.yml
```

Specify the level of verbosity by using the **-v**, **-vv** or **-vvv** option.

You can view the output of the Ansible playbook script on the command-line interface (CLI). The following output shows that the script has run successfully as 0 tasks have failed:

```
PLAY RECAP
server.idm.example.com : ok=18  changed=10  unreachable=0  failed=0  skipped=21
rescued=0  ignored=0
```

2. Locate the **ipa.csr** certificate signing request file on the controller and submit it to the external CA.
3. Place the IdM CA certificate signed by the external CA in the controller file system so that the playbook in the next step can find it.
4. Run the **ansible-playbook** command with the name of the playbook file that contains instructions for the final step of the installation, for example **install-server-step2.yml**. Specify the inventory file with the **-i** option:

```
$ ansible-playbook -v -i <path_to_inventory_directory>/host.server
<path_to_playbooks_directory>/install-server-step2.yml
```

You have installed an IdM server with an externally signed CA on your host using an Ansible playbook.

CHAPTER 3. INSTALLING AN IDENTITY MANAGEMENT REPLICA USING AN ANSIBLE PLAYBOOK

3.1. IDM REPLICA INSTALLATION USING AN ANSIBLE PLAYBOOK

The following sections describe how to configure a system as an IdM replica by using [Ansible](#). Configuring a system as an IdM replica enrolls it into an IdM domain and enables the system to use IdM services on IdM servers in the domain.

The deployment is managed by the **ipareplica** Ansible role. The role can use the autodiscovery mode for identifying the IdM servers, domain and other settings. However, if you deploy multiple replicas in a tier-like model, with different groups of replicas being deployed at different times, you must define specific servers or replicas for each group.



NOTE

Before installing an IdM replica using Ansible, ensure that you understand [Ansible](#) and IdM concepts. Ensure that you understand the following terms that are used in this chapter:

- Ansible roles
- Ansible nodes
- Ansible inventory
- Ansible tasks
- Ansible modules
- Ansible plays and playbooks

Overview

The installation consists of the following parts:

1. [Setting the parameters of the IdM replica deployment](#)
 - [Specifying the base, server and client variables for installing the IdM replica](#)
 - [Specifying the credentials for installing the IdM replica using an Ansible playbook](#)
2. [Deploying an IdM replica using an Ansible playbook](#)

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible control node.

3.2. SETTING THE PARAMETERS OF THE IDM REPLICA DEPLOYMENT

Before you deploy a target host as an IdM replica, configure the following settings:

- [Specify the base, server and client variables for installing the IdM replica.](#)
- [Specify the credentials for installing the IdM replica using an Ansible playbook.](#)

3.2.1. Specifying the base, server and client variables for installing the IdM replica

Complete this procedure to configure the inventory file for installing an IdM replica.

Procedure

1. Open the inventory file for editing. Specify the fully-qualified domain names (**FQDN**) of the hosts to become IdM replicas. The **FQDNs** must be valid DNS names:
 - Only numbers, alphabetic characters, and hyphens (-) are allowed. For example, underscores are not allowed and can cause DNS failures.
 - The host name must be all lower-case.

Example of a simple inventory hosts file with only the replicas' FQDN defined

```
[ipareplicas]
replica1.idm.example.com
replica2.idm.example.com
replica3.idm.example.com
[...]
```

If the IdM server is already deployed and the SRV records are set properly in the IdM DNS zone, the script automatically discovers all the other required values.

2. Optionally, provide additional information in the inventory file based on which of the following scenarios is closest to yours:
 - **Scenario 1**
If you want to avoid autodiscovery and have all replicas listed in the **[ipareplicas]** section use a specific IdM server, set the server in the **[ipaservers]** section of the inventory file.

Example inventory hosts file with the FQDN of the IdM server and replicas defined

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

[ipareplicas]
replica1.idm.example.com
replica2.idm.example.com
replica3.idm.example.com
[...]
```

- **Scenario 2**
Alternatively, if you want to avoid autodiscovery but want to deploy specific replicas with specific servers, set the servers for specific replicas individually in the **[ipareplicas]** section in the inventory file.

Example inventory file with a specific IdM server defined for a specific replica

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

[ipareplicas]
replica1.idm.example.com
```



```
[ipareplicas]
replica2.idm.example.com
replica3.idm.example.com ipareplica_servers=replica1.idm.example.com
```

In the example above, **replica3.idm.example.com** uses the already deployed **replica1.idm.example.com** as its replication source.

• Scenario 3

If you are deploying several replicas in one batch and time is a concern to you, multitier replica deployment can be useful for you. Define specific groups of replicas in the inventory file, for example **[ipareplicas_tier1]** and **[ipareplicas_tier2]**, and design separate plays for each group in the **install-replica.yml** playbook.

Example inventory file with replica tiers defined

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

[ipareplicas_tier1]
replica1.idm.example.com

[ipareplicas_tier2]
replica2.idm.example.com \
ipareplica_servers=replica1.idm.example.com,server.idm.example.com
```

The first entry in **ipareplica_servers** will be used. The second entry will be used as a fallback option. When using multiple tiers for deploying IdM replicas, you must have separate tasks in the playbook to first deploy replicas from tier1 and then replicas from tier2:

Example of a playbook file with different plays for different replica groups

```
---
- name: Playbook to configure IPA replicas (tier1)
  hosts: ipareplicas_tier1
  become: true

  roles:
  - role: ipareplica
    state: present

- name: Playbook to configure IPA replicas (tier2)
  hosts: ipareplicas_tier2
  become: true

  roles:
  - role: ipareplica
    state: present
```

• Scenario 4

If you want the replica to use a specified **firewalld** zone instead of the default one, you can specify it in the inventory file. This can be useful, for example, when you want to use an internal **firewalld** zone for your IdM installation instead of a public zone that is set as default.

If you do not set a custom zone, IdM will add its services to the default **firewalld** zone. The predefined default zone is **public**.



IMPORTANT

The specified **firewalld** zone must exist and be permanent.

Example of a simple inventory hosts file with a custom **firewalld** zone

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

[ipareplicas]
replica1.idm.example.com
replica2.idm.example.com
replica3.idm.example.com
[...]

[ipareplicas:vars]
ipareplica_firewalld_zone=custom zone
```

3.2.2. Specifying the credentials for installing the IdM replica using an Ansible playbook

Complete this procedure to configure the authorization for installing the IdM replica.

Procedure

1. Specify the **password of a user authorized to deploy replicas** for example the IdM **admin**.
 - Red Hat recommends using the Ansible Vault to store the password, and referencing the Vault file from the playbook file, for example **install-replica.yml**:

Example playbook file using principal from inventory file and password from an Ansible Vault file

```
- name: Playbook to configure IPA replicas
  hosts: ipareplicas
  become: true
  vars_files:
    - playbook_sensitive_data.yml

  roles:
    - role: ipareplica
      state: present
```

For details how to use Ansible Vault, see the official [Ansible Vault](#) documentation.

- Less securely, provide the credentials of **admin** directly in the inventory file. Use the **ipaadmin_password** option in the **[ipareplicas:vars]** section of the inventory file. The inventory file and the **install-replica.yml** playbook file can then look as follows:

Example inventory hosts.replica file

```
[...]
[ipareplicas:vars]
ipaadmin_password=Secret123
```

Example playbook using principal and password from inventory file

```
- name: Playbook to configure IPA replicas
  hosts: ipareplicas
  become: true

  roles:
    - role: ipareplica
      state: present
```

- Alternatively but also less securely, provide the credentials of another user authorized to deploy a replica directly in the inventory file. To specify a different authorized user, use the **ipaadmin_principal** option for the user name, and the **ipaadmin_password** option for the password. The inventory file and the **install-replica.yml** playbook file can then look as follows:

Example inventory hosts.replica file

```
[...]
[ipareplicas:vars]
ipaadmin_principal=my_admin
ipaadmin_password=my_admin_secret123
```

Example playbook using principal and password from inventory file

```
- name: Playbook to configure IPA replicas
  hosts: ipareplicas
  become: true

  roles:
    - role: ipareplica
      state: present
```

Additional resources

- For details on the options accepted by the **ipareplica** Ansible role, see the **/usr/share/ansible/roles/ipareplica/README.md** Markdown file.

3.3. DEPLOYING AN IDM REPLICA USING AN ANSIBLE PLAYBOOK

Complete this procedure to use an Ansible playbook to deploy an IdM replica.

Procedure

- To install an IdM replica using an Ansible playbook, use the **ansible-playbook** command with the name of the playbook file, for example **install-replica.yml**. Specify the inventory file with the **-i** option:

```
$ ansible-playbook -v -i <path_to_inventory_directory>/hosts.replica  
<path_to_playbooks_directory>/install-replica.yml
```

Specify the level of verbosity by using the **-v**, **-vv** or **-vvv** option.

Ansible informs you about the execution of the Ansible playbook script. The following output shows that the script has run successfully as 0 tasks have failed:

```
PLAY RECAP  
replica.idm.example.com : ok=18  changed=10  unreachable=0  failed=0  skipped=21  
rescued=0  ignored=0
```

You have now installed an IdM replica.

CHAPTER 4. INSTALLING AN IDENTITY MANAGEMENT CLIENT USING AN ANSIBLE PLAYBOOK

4.1. IDM CLIENT INSTALLATION USING AN ANSIBLE PLAYBOOK

The following sections describe how to configure a system as an Identity Management (IdM) client by using [Ansible](#). Configuring a system as an IdM client enrolls it into an IdM domain and enables the system to use IdM services on IdM servers in the domain.

The deployment is managed by the **ipaclient** Ansible role. By default, the role uses the autodiscovery mode for identifying the IdM servers, domain and other settings. The role can be modified to have the Ansible playbook use the settings specified, for example in the inventory file.



NOTE

Before installing an IdM client using Ansible, ensure that you understand [Ansible](#) and IdM concepts. Ensure that you understand the following terms that are used in this chapter:

- Ansible roles
- Ansible nodes
- Ansible inventory
- Ansible tasks
- Ansible modules
- Ansible plays and playbooks

Overview

The installation consists of the following parts:

1. [Setting the parameters of the IdM client deployment](#) to correspond to your deployment scenario:
 - Setting the parameters of the inventory file [for the autodiscovery client installation mode](#);
 - Setting the parameters of the inventory file [for when autodiscovery is not possible during client installation](#);
2. [Checking the parameters in install-client.yml](#);
3. [Deploying an IdM client using an Ansible playbook](#) ;
4. [Testing an Identity Management client after installation](#) .

The chapter also includes a section describing [how to uninstall an IdM client](#) .

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible control node.

4.2. SETTING THE PARAMETERS OF THE IDM CLIENT DEPLOYMENT

Before you deploy a target host as an IdM client, configure the [deployment instructions](#) on the control node. Additionally, configure the target host parameters depending on which of the following options you are planning:

- [Using the autodiscovery client installation mode](#)
- [Specifying the **FQDN** of the IdM server and the domain or realm information](#).

4.2.1. Setting the parameters of the inventory file for the autodiscovery client installation mode

To install an Identity Management client using an Ansible playbook, provide the following information in an inventory file, for example **inventory/hosts**:

- the information about the host
- the authorization for the task

The inventory file can be in one of many formats, depending on the inventory plugins you have. The **INI-like** format is one of Ansible's defaults and is used in the examples below.



NOTE

To use smart cards with the graphical user interface in RHEL, ensure that you include the **ipaclient_mkghomedir** variable in your Ansible playbook.

Procedure

1. Specify the fully-qualified hostname (**FQDN**) of the host to become an IdM client. The fully qualified domain name must be a valid DNS name:
 - Only numbers, alphabetic characters, and hyphens (-) are allowed. For example, underscores are not allowed and can cause DNS failures.
 - The host name must be all lower-case. No capital letters are allowed.
If the SRV records are set properly in the IdM DNS zone, the script automatically discovers all the other required values.

Example of a simple inventory hosts file with only the client FQDN defined

```
[ipaclients]
client.idm.example.com
[...]
```

2. Specify the credentials for enrolling the client. The following authentication methods are available:
 - The **password of a user authorized to enroll clients** This is the default option.
 - Red Hat recommends using the Ansible Vault to store the password, and referencing the Vault file from the playbook file, for example **install-client.yml**, directly:

Example playbook file using principal from inventory file and password from an Ansible Vault file

```

- name: Playbook to configure IPA clients with username/password
  hosts: ipaclients
  become: true
  vars_files:
  - playbook_sensitive_data.yml

  roles:
  - role: ipaclient
    state: present

```

- Less securely, provide the credentials of **admin** using the **ipaadmin_password** option in the **[ipaclients:vars]** section of the **inventory/hosts** file. Alternatively, to specify a different authorized user, use the **ipaadmin_principal** option for the user name, and the **ipaadmin_password** option for the password. The **inventory/hosts** inventory file and the **install-client.yml** playbook file can then look as follows:

Example inventory hosts file

```

[...]
[ipaclients:vars]
ipaadmin_principal=my_admin
ipaadmin_password=Secret123

```

Example Playbook using principal and password from inventory file

```

- name: Playbook to unconfigure IPA clients
  hosts: ipaclients
  become: true

  roles:
  - role: ipaclient
    state: true

```

- The **client keytab** from the previous enrollment if it is still available:
 - This option is available if the system was previously enrolled as an Identity Management client. To use this authentication method, uncomment the **#ipaclient_keytab** option, specifying the path to the file storing the keytab, for example in the **[ipaclient:vars]** section of **inventory/hosts**.
- A **random, one-time password** (OTP) to be generated during the enrollment. To use this authentication method, use the **ipaclient_use_otp=yes** option in your inventory file. For example, you can uncomment the **ipaclient_use_otp=yes** option in the **[ipaclients:vars]** section of the **inventory/hosts** file. Note that with OTP you must also specify one of the following options:
 - The **password of a user authorized to enroll clients** for example by providing a value for **ipaadmin_password** in the **[ipaclients:vars]** section of the **inventory/hosts** file.
 - The **admin keytab**, for example by providing a value for **ipaadmin_keytab** in the **[ipaclients:vars]** section of **inventory/hosts**.

Additional resources

- For details on the options accepted by the **ipaclient** Ansible role, see the **/usr/share/ansible/roles/ipaclient/README.md** README file.

4.2.2. Setting the parameters of the inventory file when autodiscovery is not possible during client installation

To install an Identity Management client using an Ansible playbook, provide the following information in an inventory file, for example **inventory/hosts**:

- the information about the host, the IdM server and the IdM domain or the IdM realm
- the authorization for the task

The inventory file can be in one of many formats, depending on the inventory plugins you have. The **INI-like** format is one of Ansible's defaults and is used in the examples below.



NOTE

To use smart cards with the graphical user interface in RHEL, ensure that you include the **ipaclient_mkhomedir** variable in your Ansible playbook.

Procedure

1. Specify the fully-qualified hostname (**FQDN**) of the host to become an IdM client. The fully qualified domain name must be a valid DNS name:
 - Only numbers, alphabetic characters, and hyphens (-) are allowed. For example, underscores are not allowed and can cause DNS failures.
 - The host name must be all lower-case. No capital letters are allowed.
2. Specify other options in the relevant sections of the **inventory/hosts** file:
 - the **FQDN** of the servers in the **[ipaservers]** section to indicate which IdM server the client will be enrolled with
 - one of the two following options:
 - the **ipaclient_domain** option in the **[ipaclients:vars]** section to indicate the DNS domain name of the IdM server the client will be enrolled with
 - the **ipaclient_realm** option in the **[ipaclients:vars]** section to indicate the name of the Kerberos realm controlled by the IdM server

Example of an inventory hosts file with the client FQDN, the server FQDN and the domain defined

```
[ipaclients]
client.idm.example.com

[ipaservers]
server.idm.example.com

[ipaclients:vars]
ipaclient_domain=idm.example.com
[...]
```


3. Specify the credentials for enrolling the client. The following authentication methods are available:

- The **password of a user authorized to enroll clients** This is the default option.
 - Red Hat recommends using the Ansible Vault to store the password, and referencing the Vault file from the playbook file, for example **install-client.yml**, directly:

Example playbook file using principal from inventory file and password from an Ansible Vault file

```
- name: Playbook to configure IPA clients with username/password
  hosts: ipaclients
  become: true
  vars_files:
  - playbook_sensitive_data.yml

  roles:
  - role: ipaclient
    state: present
```

- Less securely, provide the credentials of **admin** using the **ipaadmin_password** option in the **[ipaclients:vars]** section of the **inventory/hosts** file. Alternatively, to specify a different authorized user, use the **ipaadmin_principal** option for the user name, and the **ipaadmin_password** option for the password. The **install-client.yml** playbook file can then look as follows:

Example inventory hosts file

```
[...]
[ipaclients:vars]
ipaadmin_principal=my_admin
ipaadmin_password=Secret123
```

Example Playbook using principal and password from inventory file

```
- name: Playbook to unconfigure IPA clients
  hosts: ipaclients
  become: true

  roles:
  - role: ipaclient
    state: true
```

- The **client keytab** from the previous enrollment if it is still available:
 - This option is available if the system was previously enrolled as an Identity Management client. To use this authentication method, uncomment the **ipaclient_keytab** option, specifying the path to the file storing the keytab, for example in the **[ipaclient:vars]** section of **inventory/hosts**.
- A **random, one-time password**(OTP) to be generated during the enrollment. To use this authentication method, use the **ipaclient_use_otp=yes** option in your inventory file. For example, you can uncomment the **#ipaclient_use_otp=yes** option in the **[ipaclients:vars]**

section of the **inventory/hosts** file. Note that with OTP you must also specify one of the following options:

- The **password of a user authorized to enroll clients** for example by providing a value for **ipaadmin_password** in the **[ipaclients:vars]** section of the **inventory/hosts** file.
- The **admin keytab**, for example by providing a value for **ipaadmin_keytab** in the **[ipaclients:vars]** section of **inventory/hosts**.

Additional resources

- For details on the options accepted by the **ipaclient** Ansible role, see the **/usr/share/ansible/roles/ipaclient/README.md** README file.

4.2.3. Checking the parameters in the install-client.yml file

The **install-client.yml** playbook file contains instructions for the IdM client deployment.

- Open the file and check if the instructions in the playbook correspond to what you are planning for your deployment. The contents typically look like this:

```
---
- name: Playbook to configure IPA clients with username/password
  hosts: ipaclients
  become: true

  roles:
    - role: ipaclient
      state: present
```

This is what the individual entries mean:

- The **hosts** entry specifies the section of the **inventory/hosts** file where the ansible script searches the **FQDNs** of the hosts on which the **ipa-client-install** script shall be run.
- The **become: true** entry specifies that root's credentials will be invoked during the execution of the **ipa-client-install** script.
- The **role: ipaclient** entry specifies the role that will be installed on the host: in this case, it is the ipa client role.
- The **state: present** entry specifies that the client should be installed rather than uninstalled (**absent**).

4.2.4. Authorization options for IdM client enrollment using an Ansible playbook

This referential section presents individual authorization options for IdM client enrollment with examples of inventory and playbook files.

Table 4.1. Authorization options for IdM client enrollment using Ansible

Authorization option	Note	Example inventory file	Example install-client.yml playbook file
Password of a user authorized to enroll a client: Option 1	Password stored in Ansible vault	<pre>[ipaclients:vars] [...]</pre>	<pre>- name: Playbook to configure IPA clients with username/password hosts: ipaclients become: true vars_files: - playbook_sensitive_data.yml roles: - role: ipaclient state: present</pre>
Password of a user authorized to enroll a client: Option 2	Password stored in inventory file	<pre>[ipaclients:vars] ipaadmin_password=Secret123</pre>	<pre>- name: Playbook to configure IPA clients hosts: ipaclients become: true roles: - role: ipaclient state: true</pre>
A random, one-time password (OTP): Option 1	OTP + administrat or password	<pre>[ipaclients:vars] ipaadmin_password=Secret123 ipaclient_use_otp=yes</pre>	<pre>- name: Playbook to configure IPA clients hosts: ipaclients become: true roles: - role: ipaclient state: true</pre>
A random, one-time password (OTP): Option 2	OTP + an admin keytab	<pre>[ipaclients:vars] ipaadmin_keytab=/tmp/admin.keytab ipaclient_use_otp=yes</pre>	<pre>- name: Playbook to configure IPA clients hosts: ipaclients become: true roles: - role: ipaclient state: true</pre>

Authorization option	Note	Example inventory file	Example <code>install-client.yml</code> playbook file
The client keytab from the previous enrollment		<pre>[ipaclients:vars] ipaclient_keytab=/tmp/krb5.keytab</pre>	<pre>- name: Playbook to configure IPA clients hosts: ipaclients become: true roles: - role: ipaclient state: true</pre>

4.3. DEPLOYING AN IDM CLIENT USING AN ANSIBLE PLAYBOOK

Complete this procedure to use an Ansible playbook to deploy an IdM client in your IdM environment.

Procedure

- To install an IdM client using an Ansible playbook, use the **ansible-playbook** command with the name of the playbook file, for example **install-client.yml**. Specify the inventory file with the **-i** option:

```
$ ansible-playbook -v -i inventory/hosts install-client.yml
```

Specify the level of verbosity by using the **-v**, **-vv** or **-vvv** option.

Ansible informs you about the execution of the Ansible playbook script. The following output shows that the script has run successfully as no tasks have failed:

```
PLAY RECAP
client1.idm.example.com : ok=18 changed=10 unreachable=0 failed=0 skipped=21
rescued=0 ignored=0
```



NOTE

Ansible uses different colors to provide different types of information about the running process. You can modify the default colors in the **[colors]** section of the **/etc/ansible/ansible.cfg** file:

```
[colors]
[...]
#error = red
#debug = dark gray
#deprecate = purple
#skip = cyan
#unreachable = red
#ok = green
#changed = yellow
[...]
```

You have now installed an IdM client on your host using an Ansible playbook.

4.4. TESTING AN IDENTITY MANAGEMENT CLIENT AFTER ANSIBLE INSTALLATION

The command-line interface (CLI) informs you that the **ansible-playbook** command was successful, but you can also do your own test.

To test that the Identity Management client can obtain information about users defined on the server, check that you are able to resolve a user defined on the server. For example, to check the default **admin** user:

```
[user@client1 ~]$ id admin
uid=1254400000(admin) gid=1254400000(admins) groups=1254400000(admins)
```

To test that authentication works correctly, **su -** as another already existing IdM user:

```
[user@client1 ~]$ su - idm_user
Last login: Thu Oct 18 18:39:11 CEST 2018 from 192.168.122.1 on pts/0
[idm_user@client1 ~]$
```

4.5. UNINSTALLING AN IDM CLIENT USING AN ANSIBLE PLAYBOOK

Complete this procedure to use an Ansible playbook to uninstall your host as an IdM client.

Prerequisites

- IdM administrator credentials.

Procedure

- To uninstall the IdM client, use the **ansible-playbook** command with the name of the playbook file, for example **uninstall-client.yml**. Specify the inventory file with the **-i** option and, optionally, specify the level of verbosity by using the **-v**, **-vv** or **-vvv** options:

```
$ ansible-playbook -v -i inventory/hosts uninstall-client.yml
```



IMPORTANT

The uninstallation of the client only removes the basic IdM configuration from the host but leaves the configuration files on the host in case you decide to re-install the client. In addition, the uninstallation has the following limitations:

- It does not remove the client host entry from the IdM LDAP server. The uninstallation only unenrolls the host.
- It does not remove any services residing on the client from IdM.
- It does not remove the DNS entries for the client from the IdM server.
- It does not remove the old principals for keytabs other than **/etc/krb5.keytab**.

Note that the uninstallation does remove all certificates that were issued for the host by the IdM CA.

Additional resources

- For more information on how to remove the IdM client configuration from both the host and the IdM environment completely, see the manual procedure for [Uninstalling an IdM client](#).

CHAPTER 5. PREPARING YOUR ENVIRONMENT FOR MANAGING IDM USING ANSIBLE PLAYBOOKS

As a system administrator managing Identity Management (IdM), when working with Red Hat Ansible Engine, it is good practice to do the following:

- Create a subdirectory dedicated to Ansible playbooks in your home directory, for example **~/MyPlaybooks**.
- Copy and adapt sample Ansible playbooks from the **/usr/share/doc/ansible-freeipa/*** and **/usr/share/doc/rhel-system-roles/*** directories and subdirectories into your **~/MyPlaybooks** directory.
- Include your inventory file in your **~/MyPlaybooks** directory.

Using this practice, you can find all your playbooks in one place and you can run your playbooks without invoking root privileges.



NOTE

You only need **root** privileges on the managed nodes to execute the **ipaserver**, **ipareplica**, **ipaclient** and **ipabackup ansible-freeipa** roles. These roles require privileged access to directories and the **dnf** software package manager.

This section describes how to create the **~/MyPlaybooks** directory and configure it so that you can use it to store and run Ansible playbooks.

Prerequisites

- You have installed an IdM server on your managed nodes, **server.idm.example.com** and **replica.idm.example.com**.
- You have configured DNS and networking so you can log in to the managed nodes, **server.idm.example.com** and **replica.idm.example.com**, directly from the control node.
- You know the IdM **admin** password.

Procedure

1. Create a directory for your Ansible configuration and playbooks in your home directory:

```
$ mkdir ~/MyPlaybooks/
```

2. Change into the **~/MyPlaybooks/** directory:

```
$ cd ~/MyPlaybooks
```

3. Create the **~/MyPlaybooks/ansible.cfg** file with the following content:

```
[defaults]
inventory = /home/your_username/MyPlaybooks/inventory
```

```
[privilege_escalation]
become=True
```

4. Create the `~/MyPlaybooks/inventory` file with the following content:

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

[us]
replica.idm.example.com

[ipaserver:children]
eu
us
```

This configuration defines two host groups, **eu** and **us**, for hosts in these locations. Additionally, this configuration defines the **ipaserver** host group, which contains all hosts from the **eu** and **us** groups.

5. [Optional] Create an SSH public and private key. To simplify access in your test environment, do not set a password on the private key:

```
$ ssh-keygen
```

6. Copy the SSH public key to the IdM **admin** account on each managed node:

```
$ ssh-copy-id admin@server.idm.example.com
$ ssh-copy-id admin@replica.idm.example.com
```

These commands require that you enter the IdM **admin** password.

Additional resources

- For more information on installing an IdM server using an Ansible playbook, see [Installing an Identity Management server using an Ansible playbook](#).
- For an overview of available formats for an Ansible inventory file including examples, see [How to build your inventory](#).

CHAPTER 6. CONFIGURING GLOBAL IDM SETTINGS USING ANSIBLE PLAYBOOKS

Using the Ansible **config** module, you can retrieve and set global configuration parameters for Identity Management (IdM).

This chapter includes the following sections:

- [Retrieving IdM configuration using an Ansible playbook](#)
- [Configuring the IdM CA renewal server using an Ansible playbook](#)
- [Configuring the default shell for IdM users using an Ansible playbook](#)

6.1. RETRIEVING IDM CONFIGURATION USING AN ANSIBLE PLAYBOOK

The following procedure describes how you can use an Ansible playbook to retrieve information about the current global IdM configuration.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible controller.

Procedure

1. Create an inventory file, for example **inventory.file**, and define the IdM server from which you want to retrieve the IdM configuration in the **[ipaserver]** section. For example, to instruct Ansible to retrieve the data from **server.idm.example.com**, enter:

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

2. Open the **/usr/share/doc/ansible-freeipa/playbooks/config/retrieve-config.yml** Ansible playbook file for editing:

```
---
- name: Playbook to handle global IdM configuration
  hosts: ipaserver
  become: no
  gather_facts: no

  tasks:
    - name: Query IPA global configuration
      ipaconfig:
        ipadmin_password: Secret123
        register: serverconfig

    - debug:
        msg: "{{ serverconfig }}"
```

3. Adapt the file by changing the following:
 - The password of IdM administrator.
 - Other values, if necessary.
4. Save the file.
5. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
/usr/share/doc/ansible-freeipa/playbooks/config/retrieve-config.yml
[...]
TASK [debug]
ok: [server.idm.example.com] => {
  "msg": {
    "ansible_facts": {
      "discovered_interpreter_"
    },
    "changed": false,
    "config": {
      "ca_renewal_master_server": "server.idm.example.com",
      "configstring": [
        "AllowNThash",
        "KDC:Disable Last Success"
      ],
      "defaultgroup": "ipausers",
      "defaultshell": "/bin/bash",
      "emaildomain": "idm.example.com",
      "enable_migration": false,
      "groupsearch": [
        "cn",
        "description"
      ],
      "homedirectory": "/home",
      "maxhostname": "64",
      "maxusername": "64",
      "pac_type": [
        "MS-PAC",
        "nfs:NONE"
      ],
      "pwdexpnotify": "4",
      "searchrecordslimit": "100",
      "searchtimelimit": "2",
      "selinuxusermapdefault": "unconfined_u:s0-s0:c0.c1023",
      "selinuxusermaporder": [
        "guest_u:s0$xguest_u:s0$user_"
      ],
      "usersearch": [
        "uid",
        "givenname",
        "sn",
        "telephonenumber",
        "ou",
        "title"
      ]
    ]
  }
}
```

```

    },
    "failed": false
  }
}

```

6.2. CONFIGURING THE IDM CA RENEWAL SERVER USING AN ANSIBLE PLAYBOOK

In an Identity Management (IdM) deployment that uses an embedded certificate authority (CA), the CA renewal server maintains and renews IdM system certificates. It ensures robust IdM deployments.

For more details on the role of the IdM CA renewal server, see [Using IdM CA renewal server](#).

The following procedure describes how you can use an Ansible playbook to configure the IdM CA renewal server.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible controller.

Procedure

1. Optional: Identify the current IdM CA renewal server:

```

$ ipa config-show | grep 'CA renewal'
IPA CA renewal master: server.idm.example.com

```

2. Create an inventory file, for example **inventory.file**, and define **ipaserver** in it:

```

[ipaserver]
server.idm.example.com

```

3. Open the **/usr/share/doc/ansible-freeipa/playbooks/config/set-ca-renewal-master-server.yml** Ansible playbook file for editing:

```

---
- name: Playbook to handle global DNS configuration
  hosts: ipaserver
  become: no
  gather_facts: no

  tasks:
    - name: set ca_renewal_master_server
      ipaconfig:
        ipadmin_password: SomeADMINpassword
        ca_renewal_master_server: carenewal.idm.example.com

```

4. Adapt the file by changing:

- The password of IdM administrator set by the **ipadmin_password** variable.
- The name of the CA renewal server set by the **ca_renewal_master_server** variable.

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file and the inventory file:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file  
/usr/share/doc/ansible-freeipa/playbooks/config/set-ca-renewal-master-server.yml
```

Verification steps

You can verify that the CA renewal server has been changed:

1. Log into **ipaserver** as IdM administrator:

```
$ ssh admin@server.idm.example.com  
Password:  
[admin@server /]$
```

2. Request the identity of the IdM CA renewal server:

```
$ ipa config-show | grep 'CA renewal'  
IPA CA renewal master: carenewal.idm.example.com
```

The output shows the **carenewal.idm.example.com** server is the new CA renewal server.

6.3. CONFIGURING THE DEFAULT SHELL FOR IDM USERS USING AN ANSIBLE PLAYBOOK

The shell is a program that accepts and interprets commands. Several shells are available in Red Hat Enterprise Linux (RHEL), such as **bash**, **sh**, **ksh**, **zsh**, **fish**, and others. **Bash**, or **/bin/bash**, is a popular shell on most Linux systems, and it is normally the default shell for user accounts on RHEL.

The following procedure describes how you can use an Ansible playbook to configure **sh**, an alternative shell, as the default shell for IdM users.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible controller.

Procedure

1. Optional: Use the **retrieve-config.yml** Ansible playbook to identify the current shell for IdM users. See [Retrieving IdM configuration using an Ansible playbook](#) for details.
2. Create an inventory file, for example **inventory.file**, and define **ipaserver** in it:

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

3. Open the **/usr/share/doc/ansible-freeipa/playbooks/config/ensure-config-options-are-set.yml** Ansible playbook file for editing:

```
---
```

```
- name: Playbook to ensure some config options are set
  hosts: ipaserver
  become: true

  tasks:
    # Set defaultlogin and maxusername
    - ipaconfig:
        ipaadmin_password: Secret123
        defaultshell: /bin/bash
        maxusername: 64
```

4. Adapt the file by changing the following:

- The password of IdM administrator set by the **ipaadmin_password** variable.
- The default shell of the IdM users set by the **defaultshell** variable into **/bin/sh**.

5. Save the file.

6. Run the Ansible playbook. Specify the playbook file and the inventory file:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
/usr/share/doc/ansible-freeipa/playbooks/config/ensure-config-options-are-set.yml
```

Verification steps

You can verify that the default user shell has been changed by starting a new session in IdM:

1. Log into **ipaserver** as IdM administrator:

```
$ ssh admin@server.idm.example.com
Password:
[admin@server ~]$
```

2. Display the current shell:

```
[admin@server ~]$ echo "$SHELL"
/bin/sh
```

The logged-in user is using the **sh** shell.

Additional resources

- You can see sample Ansible playbooks for configuring global IdM settings and a list of possible variables in the **README-config.md** Markdown file available in the **/usr/share/doc/ansible-freeipa/** directory.
- You can see sample Ansible playbooks for various IdM configuration-related operations in the **/usr/share/doc/ansible-freeipa/playbooks/config** directory.

CHAPTER 7. MANAGING USER ACCOUNTS USING ANSIBLE PLAYBOOKS

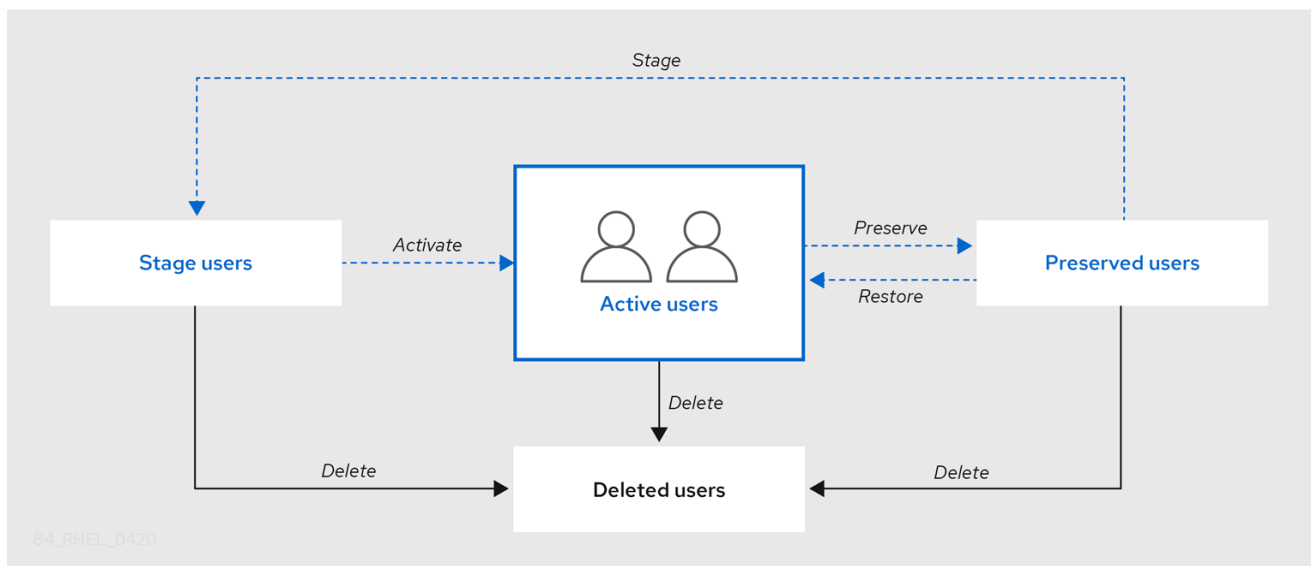
You can manage users in IdM using Ansible playbooks. After presenting the [user life cycle](#), this chapter describes how to use Ansible playbooks for the following operations:

- [Ensuring the presence of a single user](#) listed directly in the **YML** file.
- [Ensuring the presence of multiple users](#) listed directly in the **YML** file.
- [Ensuring the presence of multiple users](#) listed in a **JSON** file that is referenced from the **YML** file.
- [Ensuring the absence of users](#) listed directly in the **YML** file.

7.1. USER LIFE CYCLE

IdM (Identity Management) supports three user account states:

- **Stage** users are not allowed to authenticate. This is an initial state. Some of the user account properties required for active users cannot be set, for example, group membership.
- **Active** users are allowed to authenticate. All required user account properties must be set in this state.
- **Preserved** users are former active users that are considered inactive and cannot authenticate to IdM. Preserved users retain most of the account properties they had as active users, but they are not part of any user groups.



You can delete user entries permanently from the IdM database.



IMPORTANT

Deleted user accounts cannot be restored. When you delete a user account, all the information associated with the account is permanently lost.

A new administrator can only be created by a user with administrator rights, such as the default admin user. If you accidentally delete all administrator accounts, the Directory Manager must create a new administrator manually in the Directory Server.



WARNING

Do not delete the **admin** user. As **admin** is a pre-defined user required by IdM, this operation causes problems with certain commands. If you want to define and use an alternative admin user, disable the pre-defined **admin** user with **ipa user-disable admin** after you granted admin permissions to at least one different user.



WARNING

Do not add local users to IdM. The Name Service Switch (NSS) always resolves IdM users and groups before resolving local users and groups. This means that, for example, IdM group membership does not work for local users.

7.2. ENSURING THE PRESENCE OF AN IDM USER USING AN ANSIBLE PLAYBOOK

The following procedure describes ensuring the presence of a user in IdM using an Ansible playbook.

Prerequisites

- You know the IdM administrator password.
- The [ansible-freeipa](#) package is installed on the Ansible controller.

Procedure

1. Create an inventory file, for example **inventory.file**, and define **ipaserver** in it:

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

2. Create an Ansible playbook file with the data of the user whose presence in IdM you want to ensure. To simplify this step, you can copy and modify the example in the **/usr/share/doc/ansible-freeipa/playbooks/user/add-user.yml** file. For example, to create user named *idm_user* and add *Password123* as the user password:

```
---
- name: Playbook to handle users
  hosts: ipaserver
  become: true
```

```

tasks:
- name: Create user idm_user
  ipauser:
    ipaadmin_password: MySecret123
    name: idm_user
    first: Alice
    last: Acme
    uid: 1000111
    gid: 10011
    phone: "+555123457"
    email: idm_user@acme.com
    passwordexpiration: "2023-01-19 23:59:59"
    password: "Password123"
    update_password: on_create

```

You must use the following options to add a user:

- **name:** the login name
- **first:** the first name string
- **last:** the last name string

For the full list of available user options, see the `/usr/share/doc/ansible-freeipa/README-user.md` Markdown file.



NOTE

If you use the **update_password: on_create** option, Ansible only creates the user password when it creates the user. If the user is already created with a password, Ansible does not generate a new password.

3. Run the playbook:

```

$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
  path_to_playbooks_directory/add-IdM-user.yml

```

Verification steps

- You can verify if the new user account exists in IdM by using the **ipa user-show** command:

1. Log into **ipaserver** as admin:

```

$ ssh admin@server.idm.example.com
Password:
[admin@server /]$

```

2. Request a Kerberos ticket for admin:

```

$ kinit admin
Password for admin@IDM.EXAMPLE.COM:

```

3. Request information about *idm_user*:


```
$ ipa user-show idm_user
User login: idm_user
First name: Alice
Last name: Acme
....
```

The user named *idm_user* is present in IdM.

7.3. ENSURING THE PRESENCE OF MULTIPLE IDM USERS USING ANSIBLE PLAYBOOKS

The following procedure describes ensuring the presence of multiple users in IdM using an Ansible playbook.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible controller.

Procedure

1. Create an inventory file, for example **inventory.file**, and define **ipaserver** in it:

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

2. Create an Ansible playbook file with the data of the users whose presence you want to ensure in IdM. To simplify this step, you can copy and modify the example in the **/usr/share/doc/ansible-freeipa/playbooks/user/ensure-users-present.yml** file. For example, to create users *idm_user_1*, *idm_user_2*, and *idm_user_3*, and add *Password123* as the password of *idm_user_1*:

```
---
- name: Playbook to handle users
  hosts: ipaserver
  become: true

  tasks:
  - name: Create user idm_users
    ipauser:
      ipaadmin_password: MySecret123
      users:
      - name: idm_user_1
        first: Alice
        last: Acme
        uid: 10001
        gid: 10011
        phone: "+555123457"
        email: idm_user@acme.com
        passwordexpiration: "2023-01-19 23:59:59"
        password: "Password123"
      - name: idm_user_2
        first: Bob
        last: Acme
```

```
uid: 100011
gid: 10011
- name: idm_user_3
  first: Eve
  last: Acme
  uid: 1000111
  gid: 10011
```



NOTE

If you do not specify the **update_password: on_create** option, Ansible re-sets the user password every time the playbook is run: if the user has changed the password since the last time the playbook was run, Ansible re-sets password.

3. Run the playbook:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
path_to_playbooks_directory/add-users.yml
```

Verification steps

- You can verify if the user account exists in IdM by using the **ipa user-show** command:

1. Log into **ipaserver** as administrator:

```
$ ssh administrator@server.idm.example.com
Password:
[admin@server /]$
```

2. Display information about *idm_user_1*:

```
$ ipa user-show idm_user_1
User login: idm_user_1
First name: Alice
Last name: Acme
Password: True
....
```

The user named *idm_user_1* is present in IdM.

7.4. ENSURING THE PRESENCE OF MULTIPLE IDM USERS FROM A JSON FILE USING ANSIBLE PLAYBOOKS

The following procedure describes how you can ensure the presence of multiple users in IdM using an Ansible playbook. The users are stored in a **JSON** file.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible controller.

Procedure

1. Create an inventory file, for example **inventory.file**, and define **ipaserver** in it:

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

2. Create an Ansible playbook file with the necessary tasks. Reference the **JSON** file with the data of the users whose presence you want to ensure. To simplify this step, you can copy and modify the example in the **/usr/share/doc/ansible-freeipa/ensure-users-present.ymlfile.yml** file:

```
---
- name: Ensure users' presence
  hosts: ipaserver
  become: true

  tasks:
    - name: Include users.json
      include_vars:
        file: users.json

    - name: Users present
      ipauser:
        ipadmin_password: MySecret123
        users: "{{ users }}"
```

3. Create the **users.json** file, and add the IdM users into it. To simplify this step, you can copy and modify the example in the **/usr/share/doc/ansible-freeipa/playbooks/user/users.json** file. For example, to create users *idm_user_1*, *idm_user_2*, and *idm_user_3*, and add *Password123* as the password of *idm_user_1*:

```
{
  "users": [
    {
      "name": "idm_user_1",
      "first": "Alice",
      "last": "Acme",
      "password": "Password123"
    },
    {
      "name": "idm_user_2",
      "first": "Bob",
      "last": "Acme"
    },
    {
      "name": "idm_user_3",
      "first": "Eve",
      "last": "Acme"
    }
  ]
}
```

4. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
path_to_playbooks_directory/ensure-users-present-jsonfile.yml
```

Verification steps

- You can verify if the user accounts are present in IdM using the **ipa user-show** command:

- Log into **ipaserver** as administrator:

```
$ ssh administrator@server.idm.example.com
Password:
[admin@server /]$
```

- Display information about *idm_user_1*:

```
$ ipa user-show idm_user_1
User login: idm_user_1
First name: Alice
Last name: Acme
Password: True
....
```

The user named *idm_user_1* is present in IdM.

7.5. ENSURING THE ABSENCE OF USERS USING ANSIBLE PLAYBOOKS

The following procedure describes how you can use an Ansible playbook to ensure that specific users are absent from IdM.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible controller.

Procedure

- Create an inventory file, for example **inventory.file**, and define **ipaserver** in it:

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

- Create an Ansible playbook file with the users whose absence from IdM you want to ensure. To simplify this step, you can copy and modify the example in the **/usr/share/doc/ansible-freeipa/playbooks/user/ensure-users-present.yml** file. For example, to delete users *idm_user_1*, *idm_user_2*, and *idm_user_3*:

```
---
- name: Playbook to handle users
  hosts: ipaserver
  become: true
```

```

tasks:
- name: Delete users idm_user_1, idm_user_2, idm_user_3
  ipauser:
    ipaadmin_password: MySecret123
    users:
      - name: idm_user_1
      - name: idm_user_2
      - name: idm_user_3
    state: absent

```

3. Run the Ansible playbook specifying the playbook file and the inventory file:

```

$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
path_to_playbooks_directory/delete-users.yml

```

Verification steps

You can verify that the user accounts do not exist in IdM by using the **ipa user-show** command:

1. Log into **ipaserver** as administrator:

```

$ ssh administrator@server.idm.example.com
Password:
[admin@server /]$

```

2. Request information about *idm_user_1*:

```

$ ipa user-show idm_user_1
ipa: ERROR: idm_user_1: user not found

```

The user named *idm_user_1* does not exist in IdM.

Additional resources

- You can see sample Ansible playbooks for other IdM user-related actions such as preserving, deleting, enabling, disabling, unlocking and undeleting users in the README-user.md Markdown file available in the **/usr/share/doc/ansible-freeipa/** directory. The file also contains the definitions of **ipauser** variables.
- You can also see sample Ansible playbooks in the **/usr/share/doc/ansible-freeipa/playbooks/user** directory.

CHAPTER 8. MANAGING USER GROUPS USING ANSIBLE PLAYBOOKS

This section introduces user group management using Ansible playbooks.

A user group is a set of users with common privileges, password policies, and other characteristics.

A user group in Identity Management (IdM) can include:

- IdM users
- other IdM user groups
- external users, which are users that exist outside of IdM

The section includes the following topics:

- [The different group types in IdM](#)
- [Direct and indirect group members](#)
- [Ensuring the presence of IdM groups and group members using Ansible playbooks](#)
- [Ensuring the presence of member managers in IdM user groups using Ansible playbooks](#)
- [Ensuring the absence of member managers in IdM user groups using Ansible playbooks](#)

8.1. THE DIFFERENT GROUP TYPES IN IDM

IdM supports the following types of groups:

POSIX groups (the default)

POSIX groups support Linux POSIX attributes for their members. Note that groups that interact with Active Directory cannot use POSIX attributes.

POSIX attributes identify users as separate entities. Examples of POSIX attributes relevant to users include **uidNumber**, a user number (UID), and **gidNumber**, a group number (GID).

Non-POSIX groups

Non-POSIX groups do not support POSIX attributes. For example, these groups do not have a GID defined.

All members of this type of group must belong to the IdM domain.

External groups

Use external groups to add group members that exist in an identity store outside of the IdM domain, such as:

- A local system
- An Active Directory domain
- A directory service

External groups do not support POSIX attributes. For example, these groups do not have a GID defined.

Table 8.1. User groups created by default

Group name	Default group members
ipausers	All IdM users
admins	Users with administrative privileges, including the default admin user
editors	This is a legacy group that no longer has any special privileges
trust admins	Users with privileges to manage the Active Directory trusts

When you add a user to a user group, the user gains the privileges and policies associated with the group. For example, to grant administrative privileges to a user, add the user to the **admins** group.

**WARNING**

Do not delete the **admins** group. As **admins** is a pre-defined group required by IdM, this operation causes problems with certain commands.

In addition, IdM creates *user private groups* by default whenever a new user is created in IdM. For more information about private groups, see [Adding users without a private group](#).

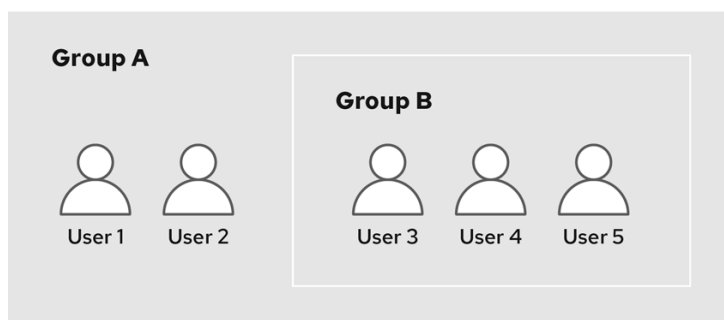
8.2. DIRECT AND INDIRECT GROUP MEMBERS

User group attributes in IdM apply to both direct and indirect members: when group B is a member of group A, all users in group B are considered indirect members of group A.

For example, in the following diagram:

- User 1 and User 2 are *direct members* of group A.
- User 3, User 4, and User 5 are *indirect members* of group A.

Figure 8.1. Direct and Indirect Group Membership



B4_RHEL_0420

If you set a password policy for user group A, the policy also applies to all users in user group B.

8.3. ENSURING THE PRESENCE OF IDM GROUPS AND GROUP MEMBERS USING ANSIBLE PLAYBOOKS

The following procedure describes ensuring the presence of IdM groups and group members – both users and user groups – using an Ansible playbook.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible controller.
- The users you want to reference in your Ansible playbook exist in IdM. For details on ensuring the presence of users using Ansible, see [Managing user accounts using Ansible playbooks](#).

Procedure

1. Create an inventory file, for example **inventory.file**, and define **ipaserver** in it:

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

2. Create an Ansible playbook file with the necessary user and group information:

```
---
- name: Playbook to handle groups
  hosts: ipaserver
  become: true

  tasks:
    - name: Create group ops with gid 1234
      ipagroup:
        ipadmin_password: MySecret123
        name: ops
        gidnumber: 1234

    - name: Create group sysops
      ipagroup:
        ipadmin_password: MySecret123
        name: sysops
        user:
          - idm_user

    - name: Create group appops
      ipagroup:
        ipadmin_password: MySecret123
        name: appops

    - name: Add group members sysops and appops to group ops
      ipagroup:
        ipadmin_password: MySecret123
        name: ops
        group:
          - sysops
          - appops
```


3. Run the playbook:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
path_to_playbooks_directory/add-group-members.yml
```

Verification steps

You can verify if the **ops** group contains **sysops** and **appops** as direct members and **idm_user** as an indirect member by using the **ipa group-show** command:

1. Log into **ipaserver** as administrator:

```
$ ssh admin@server.idm.example.com
Password:
[admin@server /]$
```

2. Display information about **ops**:

```
ipaserver]$ ipa group-show ops
Group name: ops
GID: 1234
Member groups: sysops, appops
Indirect Member users: idm_user
```

The **appops** and **sysops** groups – the latter including the **idm_user** user – exist in IdM.

Additional resources

- For more information about ensuring the presence of user groups using Ansible, see the [/usr/share/doc/ansible-freeipa/README-group.md](#) Markdown file.

8.4. ENSURING THE PRESENCE OF MEMBER MANAGERS IN IDM USER GROUPS USING ANSIBLE PLAYBOOKS

The following procedure describes ensuring the presence of IdM member managers – both users and user groups – using an Ansible playbook.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible controller.
- You must have the name of the user or group you are adding as member managers and the name of the group you want them to manage.

Procedure

1. Create an inventory file, for example **inventory.file**, and define **ipaserver** in it:

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

2. Create an Ansible playbook file with the necessary user and group member management information:

```
---
- name: Playbook to handle membership management
  hosts: ipaserver
  become: true

  tasks:
    - name: Ensure user test is present for group_a
      ipagroup:
        ipaadmin_password: MySecret123
        name: group_a
        membermanager_user: test

    - name: Ensure group_admins is present for group_a
      ipagroup:
        ipaadmin_password: MySecret123
        name: group_a
        membermanager_group: group_admins
```

3. Run the playbook:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
path_to_playbooks_directory/add-member-managers-user-groups.yml
```

Verification steps

You can verify if the **group_a** group contains **test** as a member manager and **group_admins** is a member manager of **group_a** by using the **ipa group-show** command:

1. Log into **ipaserver** as administrator:

```
$ ssh admin@server.idm.example.com
Password:
[admin@server ~]$
```

2. Display information about *managergroup1*:

```
ipaserver]$ ipa group-show group_a
Group name: group_a
GID: 1133400009
Membership managed by groups: group_admins
Membership managed by users: test
```

Additional resources

- See **ipa host-add-member-manager --help**.
- See the **ipa** man page.

8.5. ENSURING THE ABSENCE OF MEMBER MANAGERS IN IDM USER GROUPS USING ANSIBLE PLAYBOOKS

The following procedure describes ensuring the absence of IdM member managers – both users and user groups – using an Ansible playbook.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible controller.
- You must have the name of the existing member manager user or group you are removing and the name of the group they are managing.

Procedure

1. Create an inventory file, for example **inventory.file**, and define **ipaserver** in it:

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

2. Create an Ansible playbook file with the necessary user and group member management information:

```
---
- name: Playbook to handle membership management
  hosts: ipaserver
  become: true

  tasks:
  - name: Ensure member manager user and group members are absent for group_a
    ipagroup:
      ipadmin_password: MySecret123
      name: group_a
      membermanager_user: test
      membermanager_group: group_admins
      action: member
      state: absent
```

3. Run the playbook:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
path_to_playbooks_directory/ensure-member-managers-are-absent.yml
```

Verification steps

You can verify if the **group_a** group does not contain **test** as a member manager and **group_admins** as a member manager of **group_a** by using the **ipa group-show** command:

1. Log into **ipaserver** as administrator:

```
$ ssh admin@server.idm.example.com
Password:
[admin@server /]$
```

2. Display information about group_a:

■

```
ipaserver]$ ipa group-show group_a
Group name: group_a
GID: 1133400009
```

Additional resources

- See **ipa host-remove-member-manager --help**.
- See the **ipa** man page.

CHAPTER 9. USING ANSIBLE PLAYBOOKS TO MANAGE SELF-SERVICE RULES IN IDM

This section introduces self-service rules in Identity Management (IdM) and describes how to create and edit self-service access rules using Ansible playbooks. Self-service access control rules allow an IdM entity to perform specified operations on its IdM Directory Server entry.

This section covers the following topics:

- [Self-service access control in IdM](#)
- [Using Ansible to ensure that a self-service rule is present](#)
- [Using Ansible to ensure that a self-service rule is absent](#)
- [Using Ansible to ensure that a self-service rule has specific attributes](#)
- [Using Ansible to ensure that a self-service rule does not have specific attributes](#)

9.1. SELF-SERVICE ACCESS CONTROL IN IDM

Self-service access control rules define which operations an Identity Management (IdM) entity can perform on its IdM Directory Server entry: for example, IdM users have the ability to update their own passwords.

This method of control allows an authenticated IdM entity to edit specific attributes within its LDAP entry, but does not allow **add** or **delete** operations on the entire entry.



WARNING

Be careful when working with self-service access control rules: configuring access control rules improperly can inadvertently elevate an entity's privileges.

9.2. USING ANSIBLE TO ENSURE THAT A SELF-SERVICE RULE IS PRESENT

The following procedure describes how to use an Ansible playbook to define self-service rules and ensure their presence on an Identity Management (IdM) server. In this example, the new **Users can manage their own name details** rule grants users the ability to change their own **givenname**, **displayname**, **title** and **initials** attributes. This allows them to, for example, change their display name or initials if they want to.

Prerequisites

- You know the IdM administrator password.
- You have configured an Ansible control node that meets the following requirements:
 - You are using Ansible version 2.8 or later.

- You have installed the [ansible-freeipa](#) package.
- In the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server where you are configuring these options.

Procedure

1. Navigate to the `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks/
```

2. Make a copy of the **selfservice-present.yml** file located in the `/usr/share/doc/ansible-freeipa/playbooks/selfservice/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/selfservice/selfservice-present.yml  
selfservice-present-copy.yml
```

3. Open the **selfservice-present-copy.yml** Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the **ipaselfservice** task section:
 - Set the **ipaadmin_password** variable to the password of the IdM administrator.
 - Set the **name** variable to the name of the new self-service rule.
 - Set the **permission** variable to a comma-separated list of permissions to grant: **read** and **write**.
 - Set the **attribute** variable to a list of attributes that users can manage themselves: **givenname**, **displayname**, **title**, and **initials**.

This is the modified Ansible playbook file for the current example:

```
---
- name: Self-service present
  hosts: ipaserver
  become: true

  tasks:
    - name: Ensure self-service rule "Users can manage their own name details" is present
      ipaselfservice:
        ipaadmin_password: Secret123
        name: "Users can manage their own name details"
        permission: read, write
        attribute:
          - givenname
          - displayname
          - title
          - initials
```

5. Save the file.
6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i inventory selfservice-present-copy.yml
```

Additional resources

- For more information on the concept of self-service rules, see [Self-service access control in IdM](#).
- For more sample Ansible playbooks that use the **ipaselfservice** module, see:
 - The **README-selfservice.md** file available in the `/usr/share/doc/ansible-freeipa/` directory. This file also contains the definitions of the **ipaselfservice** variables.
 - The `/usr/share/doc/ansible-freeipa/playbooks/selfservice` directory.

9.3. USING ANSIBLE TO ENSURE THAT A SELF-SERVICE RULE IS ABSENT

The following procedure describes how to use an Ansible playbook to ensure a specified self-service rule is absent from your IdM configuration. The example below describes how to make sure the **Users can manage their own name details** self-service rule does not exist in IdM. This will ensure that users cannot, for example, change their own display name or initials.

Prerequisites

- You know the IdM administrator password.
- You have configured an Ansible control node that meets the following requirements:
 - You are using Ansible version 2.8 or later.
 - You have installed the [ansible-freeipa](#) package.
 - In the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server where you are configuring these options.

Procedure

1. Navigate to the `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks/
```

2. Make a copy of the **selfservice-absent.yml** file located in the `/usr/share/doc/ansible-freeipa/playbooks/selfservice/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/selfservice/selfservice-absent.yml
selfservice-absent-copy.yml
```

3. Open the **selfservice-absent-copy.yml** Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the **ipaselfservice** task section:
 - Set the **ipaadmin_password** variable to the password of the IdM administrator.
 - Set the **name** variable to the name of the self-service rule.
 - Set the **state** variable to **absent**.

This is the modified Ansible playbook file for the current example:

```
---
- name: Self-service absent
  hosts: ipaserver
  become: true

  tasks:
    - name: Ensure self-service rule "Users can manage their own name details" is absent
      ipaselfservice:
        ipadmin_password: Secret123
        name: "Users can manage their own name details"
        state: absent
```

5. Save the file.

6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i inventory selfservice-absent-copy.yml
```

Additional resources

- For more information on the concept of self-service rules, see [Self-service access control in IdM](#).
- For more sample Ansible playbooks that use the **ipaselfservice** module, see:
 - The **README-selfservice.md** file available in the `/usr/share/doc/ansible-freeipa/` directory. This file also contains the definitions of the **ipaselfservice** variables.
 - The `/usr/share/doc/ansible-freeipa/playbooks/selfservice` directory.

9.4. USING ANSIBLE TO ENSURE THAT A SELF-SERVICE RULE HAS SPECIFIC ATTRIBUTES

The following procedure describes how to use an Ansible playbook to ensure that an already existing self-service rule has specific settings. In the example, you ensure the **Users can manage their own name details** self-service rule also has the **surname** member attribute.

Prerequisites

- You know the IdM administrator password.
- You have configured an Ansible control node that meets the following requirements:
 - You are using Ansible version 2.8 or later.
 - You have installed the [ansible-freeipa](#) package.
 - In the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server where you are configuring these options.
- The **Users can manage their own name details** self-service rule exists in IdM.

Procedure

1. Navigate to the `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks/
```

2. Make a copy of the **selfservice-member-present.yml** file located in the `/usr/share/doc/ansible-freeipa/playbooks/selfservice/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/selfservice/selfservice-member-present.yml selfservice-member-present-copy.yml
```

3. Open the **selfservice-member-present-copy.yml** Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the **ipaselfservice** task section:
 - Set the **ipaadmin_password** variable to the password of the IdM administrator.
 - Set the **name** variable to the name of the self-service rule to modify.
 - Set the **attribute** variable to **surname**.
 - Set the **action** variable to **member**.

This is the modified Ansible playbook file for the current example:

```
---
- name: Self-service member present
  hosts: ipaserver
  become: true

  tasks:
    - name: Ensure selfservice "Users can manage their own name details" member attribute surname is present
      ipaselfservice:
        ipaadmin_password: Secret123
        name: "Users can manage their own name details"
        attribute:
          - surname
        action: member
```

5. Save the file.
6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i inventory selfservice-member-present-copy.yml
```

Additional resources

- For more information on the concept of self-service rules, see [Self-service access control in IdM](#).
- For more sample Ansible playbooks that use the **ipaselfservice** module, see:

- The **README-selfservice.md** file available in the `/usr/share/doc/ansible-freeipa/` directory. This file also contains the definitions of the **ipaselfservice** variables.
- The `/usr/share/doc/ansible-freeipa/playbooks/selfservice` directory.

9.5. USING ANSIBLE TO ENSURE THAT A SELF-SERVICE RULE DOES NOT HAVE SPECIFIC ATTRIBUTES

The following procedure describes how to use an Ansible playbook to ensure that a self-service rule does not have specific settings. You can use this playbook to make sure a self-service rule does not grant undesired access. In the example, you ensure the **Users can manage their own name details** self-service rule does not have the **givenname** and **surname** member attributes.

Prerequisites

- You know the IdM administrator password.
- You have configured an Ansible control node that meets the following requirements:
 - You are using Ansible version 2.8 or later.
 - You have installed the [ansible-freeipa](#) package.
 - In the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server where you are configuring these options.
- The **Users can manage their own name details** self-service rule exists in IdM.

Procedure

1. Navigate to the `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks/
```

2. Make a copy of the **selfservice-member-absent.yml** file located in the `/usr/share/doc/ansible-freeipa/playbooks/selfservice/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/selfservice/selfservice-member-absent.yml selfservice-member-absent-copy.yml
```

3. Open the **selfservice-member-absent-copy.yml** Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the **ipaselfservice** task section:
 - Set the **ipaadmin_password** variable to the password of the IdM administrator.
 - Set the **name** variable to the name of the self-service rule you want to modify.
 - Set the **attribute** variable to **givenname** and **surname**.
 - Set the **action** variable to **member**.
 - Set the **state** variable to **absent**.

This is the modified Ansible playbook file for the current example:

```

---
- name: Self-service member absent
  hosts: ipaserver
  become: true

  tasks:
    - name: Ensure selfservice "Users can manage their own name details" member attributes
      givenname and surname are absent
      ipaselfservice:
        ipadmin_password: Secret123
        name: "Users can manage their own name details"
        attribute:
          - givenname
          - surname
        action: member
        state: absent

```

5. Save the file.

6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i inventory selfservice-member-absent-copy.yml
```

Additional resources

- For more information on the concept of self-service rules, see [Self-service access control in IdM](#).
- For more sample Ansible playbooks that use the **ipaselfservice** module, see:
 - The **README-selfservice.md** file available in the `/usr/share/doc/ansible-freeipa/` directory. This file also contains the definitions of the **ipaselfservice** variables.
 - The `/usr/share/doc/ansible-freeipa/playbooks/selfservice` directory.

CHAPTER 10. DELEGATING PERMISSIONS TO USER GROUPS TO MANAGE USERS USING ANSIBLE PLAYBOOKS

Delegation is one of the access control methods in IdM, along with self-service rules and role-based access control (RBAC). You can use delegation to assign permissions to one group of users to manage entries for another group of users.

This section covers the following topics:

- [Delegation rules](#)
- [Creating the Ansible inventory file for IdM](#)
- [Using Ansible to ensure that a delegation rule is present](#)
- [Using Ansible to ensure that a delegation rule is absent](#)
- [Using Ansible to ensure that a delegation rule has specific attributes](#)
- [Using Ansible to ensure that a delegation rule does not have specific attributes](#)

10.1. DELEGATION RULES

You can delegate permissions to user groups to manage users by creating **delegation rules**.

Delegation rules allow a specific user group to perform write (edit) operations on specific attributes for users in another user group. This form of access control rule is limited to editing the values of a subset of attributes you specify in a delegation rule; it does not grant the ability to add or remove whole entries or control over unspecified attributes.

Delegation rules grant permissions to existing user groups in IdM. You can use delegation to, for example, allow the **managers** user group to manage selected attributes of users in the **employees** user group.

10.2. CREATING AN ANSIBLE INVENTORY FILE FOR IDM

When working with Ansible, it is good practice to create, in your home directory, a subdirectory dedicated to Ansible playbooks that you copy and adapt from the **/usr/share/doc/ansible-freeipa/*** and **/usr/share/doc/rhel-system-roles/*** subdirectories. This practice has the following advantages:

- You can find all your playbooks in one place.
- You can run your playbooks without invoking **root** privileges.

Procedure

1. Create a directory for your Ansible configuration and playbooks in your home directory:

```
$ mkdir ~/MyPlaybooks/
```

2. Change into the **~/MyPlaybooks/** directory:

```
$ cd ~/MyPlaybooks
```

3. Create the `~/MyPlaybooks/ansible.cfg` file with the following content:

```
[defaults]
inventory = /home/<username>/MyPlaybooks/inventory

[privilege_escalation]
become=True
```

4. Create the `~/MyPlaybooks/inventory` file with the following content:

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

[us]
replica.idm.example.com

[ipaserver:children]
eu
us
```

This configuration defines two host groups, **eu** and **us**, for hosts in these locations. Additionally, this configuration defines the **ipaserver** host group, which contains all hosts from the **eu** and **us** groups.

10.3. USING ANSIBLE TO ENSURE THAT A DELEGATION RULE IS PRESENT

The following procedure describes how to use an Ansible playbook to define privileges for a new IdM delegation rule and ensure its presence. In the example, the new **basic manager attributes** delegation rule grants the **managers** group the ability to read and write the following attributes for members of the **employees** group:

- **businesscategory**
- **departmentnumber**
- **employeenumber**
- **employeetype**

Prerequisites

- You know the IdM administrator password.
- You have configured an Ansible control node that meets the following requirements:
 - You are using Ansible version 2.8 or later.
 - You have installed the [ansible-freeipa](#) package.
 - You have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server where you are configuring these options.
 - Your Ansible inventory file is located in the `~/MyPlaybooks/` directory.

Procedure

1. Navigate to the `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks/
```

2. Make a copy of the `delegation-present.yml` file located in the `/usr/share/doc/ansible-freeipa/playbooks/delegation/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/delegation/delegation-present.yml
delegation-present-copy.yml
```

3. Open the `delegation-present-copy.yml` Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the `ipadelegation` task section:
 - Set the `ipaadmin_password` variable to the password of the IdM administrator.
 - Set the `name` variable to the name of the new delegation rule.
 - Set the `permission` variable to a comma-separated list of permissions to grant: `read` and `write`.
 - Set the `attribute` variable to a list of attributes the delegated user group can manage: `businesscategory`, `departmentnumber`, `employeenumber`, and `employeeetype`.
 - Set the `group` variable to the name of the group that is being given access to view or modify attributes.
 - Set the `membergroup` variable to the name of the group whose attributes can be viewed or modified.

This is the modified Ansible playbook file for the current example:

```
---
- name: Playbook to manage a delegation rule
  hosts: ipaserver
  become: true

  tasks:
  - name: Ensure delegation "basic manager attributes" is present
    ipadelegation:
      ipaadmin_password: Secret123
      name: "basic manager attributes"
      permission: read, write
      attribute:
        - businesscategory
        - departmentnumber
        - employeenumber
        - employeeetype
      group: managers
      membergroup: employees
```

5. Save the file.
6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i ~/MyPlaybooks/inventory delegation-present-copy.yml
```

Additional resources

- For more information on the concept of a delegation rule, see [Delegation rules](#).
- For more sample Ansible playbooks that use the **ipadelegation** module, see:
 - The **README-delegation.md** file available in the `/usr/share/doc/ansible-freeipa/` directory. This file also contains the definitions of the **ipadelegation** variables.
 - The `/usr/share/doc/ansible-freeipa/playbooks/ipadelegation` directory.

10.4. USING ANSIBLE TO ENSURE THAT A DELEGATION RULE IS ABSENT

The following procedure describes how to use an Ansible playbook to ensure a specified delegation rule is absent from your IdM configuration. The example below describes how to make sure the custom **basic manager attributes** delegation rule does not exist in IdM.

Prerequisites

- You know the IdM administrator password.
- You have configured an Ansible control node that meets the following requirements:
 - You are using Ansible version 2.8 or later.
 - You have installed the [ansible-freeipa](#) package.
 - You have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server where you are configuring these options.
 - Your Ansible inventory file is located in the `~/MyPlaybooks/` directory.

Procedure

1. Navigate to the `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks>/
```

2. Make a copy of the **delegation-absent.yml** file located in the `/usr/share/doc/ansible-freeipa/playbooks/delegation/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/delegation/delegation-present.yml  
delegation-absent-copy.yml
```

3. Open the **delegation-absent-copy.yml** Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the **ipadelegation** task section:
 - Set the **ipaadmin_password** variable to the password of the IdM administrator.
 - Set the **name** variable to the name of the delegation rule.

- Set the **state** variable to **absent**.

This is the modified Ansible playbook file for the current example:

```
---
- name: Delegation absent
  hosts: ipaserver
  become: true

  tasks:
    - name: Ensure delegation "basic manager attributes" is absent
      ipadelegation:
        ipadmin_password: Secret123
        name: "basic manager attributes"
        state: absent
```

5. Save the file.
6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i ~/MyPlaybooks/inventory delegation-absent-copy.yml
```

Additional resources

- For more information on the concept of a delegation rule, see [Delegation rules](#).
- For more sample Ansible playbooks that use the **ipadelegation** module, see:
 - The **README-delegation.md** file available in the `/usr/share/doc/ansible-freeipa/` directory. This file also contains the definitions of the **ipadelegation** variables.
 - The `/usr/share/doc/ansible-freeipa/playbooks/ipadelegation` directory.

10.5. USING ANSIBLE TO ENSURE THAT A DELEGATION RULE HAS SPECIFIC ATTRIBUTES

The following procedure describes how to use an Ansible playbook to ensure that a delegation rule has specific settings. You can use this playbook to modify a delegation role you have previously created. In the example, you ensure the **basic manager attributes** delegation rule only has the **departmentnumber** member attribute.

Prerequisites

- You know the IdM administrator password.
- You have configured an Ansible control node that meets the following requirements:
 - You are using Ansible version 2.8 or later.
 - You have installed the [ansible-freeipa](#) package.
 - You have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server where you are configuring these options.
 - Your Ansible inventory file is located in the `~/MyPlaybooks/` directory.

- The **basic manager attributes** delegation rule exists in IdM.

Procedure

1. Navigate to the `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks/
```

2. Make a copy of the **delegation-member-present.yml** file located in the `/usr/share/doc/ansible-freeipa/playbooks/delegation/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/delegation/delegation-member-present.yml delegation-member-present-copy.yml
```

3. Open the **delegation-member-present-copy.yml** Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the **ipadelegation** task section:
 - Set the **ipaadmin_password** variable to the password of the IdM administrator.
 - Set the **name** variable to the name of the delegation rule to modify.
 - Set the **attribute** variable to **departmentnumber**.
 - Set the **action** variable to **member**.

This is the modified Ansible playbook file for the current example:

```
---
- name: Delegation member present
  hosts: ipaserver
  become: true

  tasks:
    - name: Ensure delegation "basic manager attributes" member attribute departmentnumber
      is present
      ipadelegation:
        ipaadmin_password: Secret123
        name: "basic manager attributes"
        attribute:
          - departmentnumber
        action: member
```

5. Save the file.
6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i ~/MyPlaybooks/inventory delegation-member-present-copy.yml
```

Additional resources

- For more information on the concept of a delegation rule, see [Delegation rules](#).

- For more sample Ansible playbooks that use the **ipadelegation** module, see:
 - The **README-delegation.md** file available in the `/usr/share/doc/ansible-freeipa/` directory. This file also contains the definitions of the **ipadelegation** variables.
 - The `/usr/share/doc/ansible-freeipa/playbooks/ipadelegation` directory.

10.6. USING ANSIBLE TO ENSURE THAT A DELEGATION RULE DOES NOT HAVE SPECIFIC ATTRIBUTES

The following procedure describes how to use an Ansible playbook to ensure that a delegation rule does not have specific settings. You can use this playbook to make sure a delegation role does not grant undesired access. In the example, you ensure the **basic manager attributes** delegation rule does not have the **employeenumber** and **employeetype** member attributes.

Prerequisites

- You know the IdM administrator password.
- You have configured an Ansible control node that meets the following requirements:
 - You are using Ansible version 2.8 or later.
 - You have installed the [ansible-freeipa](#) package.
 - You have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server where you are configuring these options.
 - Your Ansible inventory file is located in the `~/MyPlaybooks/` directory.
- The **basic manager attributes** delegation rule exists in IdM.

Procedure

1. Navigate to the `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks/
```

2. Make a copy of the **delegation-member-absent.yml** file located in the `/usr/share/doc/ansible-freeipa/playbooks/delegation/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/delegation/delegation-member-absent.yml delegation-member-absent-copy.yml
```

3. Open the **delegation-member-absent-copy.yml** Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the **ipadelegation** task section:
 - Set the **ipaadmin_password** variable to the password of the IdM administrator.
 - Set the **name** variable to the name of the delegation rule to modify.
 - Set the **attribute** variable to **employeenumber** and **employeetype**.
 - Set the **action** variable to **member**.

- Set the **state** variable to **absent**.

This is the modified Ansible playbook file for the current example:

```
---
- name: Delegation member absent
  hosts: ipaserver
  become: true

  tasks:
    - name: Ensure delegation "basic manager attributes" member attributes employeenumber
      and employeetype are absent
      ipadelegation:
        ipadmin_password: Secret123
        name: "basic manager attributes"
        attribute:
          - employeenumber
          - employeetype
        action: member
        state: absent
```

5. Save the file.
6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i ~/MyPlaybooks/inventory delegation-member-absent-copy.yml
```

Additional resources

- For more information on the concept of a delegation rule, see [Delegation rules](#).
- For more sample Ansible playbooks that use the **ipadelegation** module, see:
 - The **README-delegation.md** file available in the `/usr/share/doc/ansible-freeipa/` directory. This file also contains the definitions of the **ipadelegation** variables.
 - The `/usr/share/doc/ansible-freeipa/playbooks/ipadelegation` directory.

CHAPTER 11. USING ANSIBLE PLAYBOOKS TO MANAGE ROLE-BASED ACCESS CONTROL IN IDM

Role-based access control (RBAC) is a policy-neutral access-control mechanism defined around roles and privileges. The components of RBAC in Identity Management (IdM) are roles, privileges and permissions:

- **Permissions** grant the right to perform a specific task such as adding or deleting users, modifying a group, enabling read-access, etc.
- **Privileges** combine permissions, for example all the permissions needed to add a new user.
- **Roles** grant a set of privileges to users, user groups, hosts or host groups.

Especially in large companies, using RBAC can help create a hierarchical system of administrators with their individual areas of responsibility.

This chapter describes the following operations performed when managing RBAC using Ansible playbooks:

- [Permissions in IdM](#)
- [Default managed permissions](#)
- [Privileges in IdM](#)
- [Roles in IdM](#)
- [Predefined roles in IdM](#)
- [Using Ansible to ensure an IdM RBAC role with privileges is present](#)
- [Using Ansible to ensure an IdM RBAC role is absent](#)
- [Using Ansible to ensure that a group of users is assigned to an IdM RBAC role](#)
- [Using Ansible to ensure that specific users are not assigned to an IdM RBAC role](#)
- [Using Ansible to ensure a service is a member of an IdM RBAC role](#)
- [Using Ansible to ensure a host is a member of an IdM RBAC role](#)
- [Using Ansible to ensure a host group is a member of an IdM RBAC role](#)

11.1. PERMISSIONS IN IDM

Permissions are the lowest level unit of role-based access control, they define operations together with the LDAP entries to which those operations apply. Comparable to building blocks, permissions can be assigned to as many privileges as needed.

One or more **rights** define what operations are allowed:

- **write**
- **read**
- **search**

- **compare**
- **add**
- **delete**
- **all**

These operations apply to three basic **targets**:

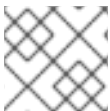
- **subtree**: a domain name (DN); the subtree under this DN
- **target filter**: an LDAP filter
- **target**: DN with possible wildcards to specify entries

Additionally, the following convenience options set the corresponding attribute(s):

- **type**: a type of object (user, group, etc); sets **subtree** and **target filter**
- **memberof**: members of a group; sets a **target filter**
- **targetgroup**: grants access to modify a specific group (such as granting the rights to manage group membership); sets a **target**

With IdM permissions, you can control which users have access to which objects and even which attributes of these objects. IdM enables you to allow or block individual attributes or change the entire visibility of a specific IdM function, such as users, groups, or sudo, to all anonymous users, all authenticated users, or just a certain group of privileged users.

For example, the flexibility of this approach to permissions is useful for an administrator who wants to limit access of users or groups only to the specific sections these users or groups need to access and to make the other sections completely hidden to them.



NOTE

A permission cannot contain other permissions.

11.2. DEFAULT MANAGED PERMISSIONS

Managed permissions are permissions that come by default with IdM. They behave like other permissions created by the user, with the following differences:

- You cannot delete them or modify their name, location, and target attributes.
- They have three sets of attributes:
 - **Default** attributes, the user cannot modify them, as they are managed by IdM
 - **Included** attributes, which are additional attributes added by the user
 - **Excluded** attributes, which are attributes removed by the user

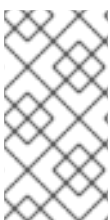
A managed permission applies to all attributes that appear in the default and included attribute sets but not in the excluded set.

**NOTE**

While you cannot delete a managed permission, setting its bind type to permission and removing the managed permission from all privileges effectively disables it.

Names of all managed permissions start with **System:**, for example **System: Add Sudo rule** or **System: Modify Services**. Earlier versions of IdM used a different scheme for default permissions. For example, the user could not delete them and was only able to assign them to privileges. Most of these default permissions have been turned into managed permissions, however, the following permissions still use the previous scheme:

- Add Automember Rebuild Membership Task
- Add Configuration Sub-Entries
- Add Replication Agreements
- Certificate Remove Hold
- Get Certificates status from the CA
- Read DNA Range
- Modify DNA Range
- Read PassSync Managers Configuration
- Modify PassSync Managers Configuration
- Read Replication Agreements
- Modify Replication Agreements
- Remove Replication Agreements
- Read LDBM Database Configuration
- Request Certificate
- Request Certificate ignoring CA ACLs
- Request Certificates from a different host
- Retrieve Certificates from the CA
- Revoke Certificate
- Write IPA Configuration

**NOTE**

If you attempt to modify a managed permission from the command line, the system does not allow you to change the attributes that you cannot modify, the command fails. If you attempt to modify a managed permission from the Web UI, the attributes that you cannot modify are disabled.

11.3. PRIVILEGES IN IDM

A privilege is a group of permissions applicable to a role.

While a permission provides the rights to do a single operation, there are certain IdM tasks that require multiple permissions to succeed. Therefore, a privilege combines the different permissions required to perform a specific task.

For example, setting up an account for a new IdM user requires the following permissions:

- Creating a new user entry
- Resetting a user password
- Adding the new user to the default IPA users group

Combining these three low-level tasks into a higher level task in the form of a custom privilege named, for example, **Add User** makes it easier for a system administrator to manage roles. IdM already contains several default privileges. Apart from users and user groups, privileges are also assigned to hosts and host groups, as well as network services. This practice permits a fine-grained control of operations by a set of users on a set of hosts using specific network services.



NOTE

A privilege may not contain other privileges.

11.4. ROLES IN IDM

A role is a list of privileges that users specified for the role possess.

In effect, permissions grant the ability to perform given low-level tasks (create a user entry, add an entry to a group, etc.), privileges combine one or more of these permissions needed for a higher-level task (such as creating a new user in a given group). Roles gather privileges together as needed: for example, a User Administrator role would be able to add, modify, and delete users.



IMPORTANT

Roles are used to classify permitted actions. They are not used as a tool to implement privilege separation or to protect from privilege escalation.



NOTE

Roles can not contain other roles.

11.5. PREDEFINED ROLES IN IDENTITY MANAGEMENT

Red Hat Identity Management provides the following range of pre-defined roles:

Table 11.1. Predefined Roles in Identity Management

Role	Privilege	Description
Helpdesk	Modify Users and Reset passwords, Modify Group membership	Responsible for performing simple user administration tasks

Role	Privilege	Description
IT Security Specialist	Netgroups Administrators, HBAC Administrator, Sudo Administrator	Responsible for managing security policy such as host-based access controls, sudo rules
IT Specialist	Host Administrators, Host Group Administrators, Service Administrators, Automount Administrators	Responsible for managing hosts
Security Architect	Delegation Administrator, Replication Administrators, Write IPA Configuration, Password Policy Administrator	Responsible for managing the Identity Management environment, creating trusts, creating replication agreements
User Administrator	User Administrators, Group Administrators, Stage User Administrators	Responsible for creating users and groups

11.6. USING ANSIBLE TO ENSURE AN IDM RBAC ROLE WITH PRIVILEGES IS PRESENT

To exercise more granular control over role-based access (RBAC) to resources in Identity Management (IdM) than the default roles provide, create a custom role.

The following procedure describes how to use an Ansible playbook to define privileges for a new IdM custom role and ensure its presence. In the example, the new **user_and_host_administrator** role contains a unique combination of the following privileges that are present in IdM by default:

- **Group Administrators**
- **User Administrators**
- **Stage User Administrators**
- **Group Administrators**

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible control node.
- You have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server on which you want to do the configuring.
- Your Ansible inventory file is located in the `~/<MyPlaybooks>/` directory.

Procedure

1. Navigate to the `~/<MyPlaybooks>/` directory:


```
$ cd ~/<MyPlaybooks>/
```

2. Make a copy of the **role-member-user-present.yml** file located in the **/usr/share/doc/ansible-freeipa/playbooks/role/** directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/role/role-member-user-present.yml role-member-user-present-copy.yml
```

3. Open the **role-member-user-present-copy.yml** Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the **iparole** task section:
 - Set the **ipaadmin_password** variable to the password of the IdM administrator.
 - Set the **name** variable to the name of the new role.
 - Set the **privilege** list to the names of the IdM privileges that you want to include in the new role.
 - Optionally, set the **user** variable to the name of the user to whom you want to grant the new role.
 - Optionally, set the **group** variable to the name of the group to which you want to grant the new role.

This is the modified Ansible playbook file for the current example:

```
---
- name: Playbook to manage IPA role with members.
  hosts: ipaserver
  become: yes
  gather_facts: no

  tasks:
  - iparole:
      ipaadmin_password: Secret123
      name: user_and_host_administrator
      user: idm_user01
      group: idm_group01
      privilege:
        - Group Administrators
        - User Administrators
        - Stage User Administrators
        - Group Administrators
```

5. Save the file.
6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i ~/<MyPlaybooks>/inventory role-member-user-present-copy.yml
```

- For more information on how to use Ansible Vault to store a password in a separate file or to encrypt it as a variable in the playbook file, see [Encrypting content with Ansible Vault](#).
- For more information on the concept of role in IdM, see [Roles in IdM](#).
- For more sample Ansible playbooks that use the **iparole** module, see:
 - The **README-role** file available in the `/usr/share/doc/ansible-freeipa/` directory. This file also contains the definitions of the **iparole** variables.
 - The `/usr/share/doc/ansible-freeipa/playbooks/iparole` directory.

11.7. USING ANSIBLE TO ENSURE AN IDM RBAC ROLE IS ABSENT

As a system administrator managing role-based access control (RBAC) in Identity Management (IdM), you may want to ensure the absence of an obsolete role so that no administrator assigns it to any user accidentally.

The following procedure describes how to use an Ansible playbook to ensure a role is absent. The example below describes how to make sure the custom **user_and_host_administrator** role does not exist in IdM.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible control node.
- You have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server on which you want to do the configuring.
- Your Ansible inventory file is located in the `~/<MyPlaybooks>/` directory.

Procedure

1. Navigate to the `~/<MyPlaybooks>/` directory:

```
$ cd ~/<MyPlaybooks>/
```

2. Make a copy of the **role-is-absent.yml** file located in the `/usr/share/doc/ansible-freeipa/playbooks/role/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/role/role-is-absent.yml role-is-absent-copy.yml
```

3. Open the **role-is-absent-copy.yml** Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the **iparole** task section:
 - Set the **ipaadmin_password** variable to the password of the IdM administrator.
 - Set the **name** variable to the name of the role.
 - Ensure that the **state** variable is set to **absent**.

This is the modified Ansible playbook file for the current example:

```

---
- name: Playbook to manage IPA role with members.
  hosts: ipaserver
  become: yes
  gather_facts: no

  tasks:
  - iparole:
    ipaadmin_password: Secret123
    name: user_and_host_administrator
    state: absent

```

5. Save the file.

6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i ~/<MyPlaybooks>/inventory role-is-absent-copy.yml
```

Additional resources

- For more information on how to use Ansible Vault to store a password in a separate file or to encrypt it as a variable in the playbook file, see [Encrypting content with Ansible Vault](#).
- For more information on the concept of role in IdM, see [Roles in IdM](#).
- For more sample Ansible playbooks that use the **iparole** module, see:
 - The **README-role** Markdown file available in the `/usr/share/doc/ansible-freeipa/` directory. This file also contains the definitions of the **iparole** variables.
 - The `/usr/share/doc/ansible-freeipa/playbooks/iparole` directory.

11.8. USING ANSIBLE TO ENSURE THAT A GROUP OF USERS IS ASSIGNED TO AN IDM RBAC ROLE

As a system administrator managing role-based access control (RBAC) in Identity Management (IdM), you may want to assign a role to a specific group of users, for example junior administrators.

The following example describes how to use an Ansible playbook to ensure the built-in IdM RBAC **helpdesk** role is assigned to **junior_sysadmins**.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible control node.
- You have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server on which you want to do the configuring.
- Your Ansible inventory file is located in the `~/<MyPlaybooks>/` directory.

Procedure

1. Navigate to the `~/<MyPlaybooks>/` directory:

```
$ cd ~/<MyPlaybooks>/
```

2. Make a copy of the **role-member-group-present.yml** file located in the `/usr/share/doc/ansible-freeipa/playbooks/role/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/role/role-member-group-present.yml  
role-member-group-present-copy.yml
```

3. Open the **role-member-group-present-copy.yml** Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the **iparole** task section:
 - Set the **ipaadmin_password** variable to the password of the IdM administrator.
 - Set the **name** variable to the name of the role you want to assign.
 - Set the **group** variable to the name of the group.
 - Set the **action** variable to **member**.

This is the modified Ansible playbook file for the current example:

```
---  
- name: Playbook to manage IPA role with members.  
  hosts: ipaserver  
  become: yes  
  gather_facts: no  
  
  tasks:  
  - iparole:  
    ipaadmin_password: Secret123  
    name: helpdesk  
    group: junior_sysadmins  
    action: member
```

5. Save the file.
6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i ~/<MyPlaybooks>/inventory role-member-group-present-  
copy.yml
```

Additional resources

- For more information on how to use Ansible Vault to store a password in a separate file or to encrypt it as a variable in the playbook file, see [Encrypting content with Ansible Vault](#).
- For more information on the concept of role in IdM, see [Roles in IdM](#).
- For more sample Ansible playbooks that use the **iparole** module, see the **README-role** Markdown file available in the `/usr/share/doc/ansible-freeipa/` directory. The file also contains the definitions of the **iparole** variables.

- For more sample Ansible playbooks that use the **iparole** module, see the `/usr/share/doc/ansible-freeipa/playbooks/iparole` directory.

11.9. USING ANSIBLE TO ENSURE THAT SPECIFIC USERS ARE NOT ASSIGNED TO AN IDM RBAC ROLE

As a system administrator managing role-based access control (RBAC) in Identity Management (IdM), you may want to ensure that an RBAC role is not assigned to specific users after they have, for example, moved to different positions within the company.

The following procedure describes how to use an Ansible playbook to ensure that the users named **user_01** and **user_02** are not assigned to the **helpdesk** role.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible control node.
- You have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server on which you want to do the configuring.
- Your Ansible inventory file is located in the `~/<MyPlaybooks>/` directory.

Procedure

1. Navigate to the `~/<MyPlaybooks>/` directory:

```
$ cd ~/<MyPlaybooks>/
```

2. Make a copy of the **role-member-user-absent.yml** file located in the `/usr/share/doc/ansible-freeipa/playbooks/role/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/role/role-member-user-absent.yml role-member-user-absent-copy.yml
```

3. Open the **role-member-user-absent-copy.yml** Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the **iparole** task section:
 - Set the **ipaadmin_password** variable to the password of the IdM administrator.
 - Set the **name** variable to the name of the role you want to assign.
 - Set the **user** list to the names of the users.
 - Set the **action** variable to **member**.
 - Set the **state** variable to **absent**.

This is the modified Ansible playbook file for the current example:

```
---
- name: Playbook to manage IPA role with members.
  hosts: ipaserver
```

```

become: yes
gather_facts: no

tasks:
- iparole:
    ipadmin_password: Secret123
    name: helpdesk
    user
    - user_01
    - user_02
    action: member
    state: absent

```

5. Save the file.

6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i ~/<MyPlaybooks>/inventory role-member-user-absent-copy.yml
```

Additional resources

- For more information on how to use Ansible Vault to store a password in a separate file or to encrypt it as a variable in the playbook file, see [Encrypting content with Ansible Vault](#).
- For more information on the concept of role in IdM, see [Roles in IdM](#).
- For more sample Ansible playbooks that use the **iparole** module, see the **README-role** Markdown file available in the **/usr/share/doc/ansible-freeipa/** directory. The file also contains the definitions of the **iparole** variables.
- For more sample Ansible playbooks that use the **iparole** module, see the **/usr/share/doc/ansible-freeipa/playbooks/iparole** directory.

11.10. USING ANSIBLE TO ENSURE A SERVICE IS A MEMBER OF AN IDM RBAC ROLE

As a system administrator managing role-based access control (RBAC) in Identity Management (IdM), you may want to ensure that a specific service that is enrolled into IdM is a member of a particular role. The following example describes how to ensure that the custom **web_administrator** role can manage the **HTTP** service that is running on the **client01.idm.example.com** server.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible control node.
- You have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server on which you want to do the configuring.
- Your Ansible inventory file is located in the **~/<MyPlaybooks>/** directory.
- The **web_administrator** role exists in IdM.

- The **HTTP/client01.idm.example.com@IDM.EXAMPLE.COM** service exists in IdM.

Procedure

1. Navigate to the `~/<MyPlaybooks>/` directory:

```
$ cd ~/<MyPlaybooks>/
```

2. Make a copy of the **role-member-service-present.yml** file located in the `/usr/share/doc/ansible-freeipa/playbooks/role/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/role/role-member-service-present-absent.yml role-member-service-present-copy.yml
```

3. Open the **role-member-service-present-copy.yml** Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the **iparole** task section:

- Set the **ipaadmin_password** variable to the password of the IdM administrator.
- Set the **name** variable to the name of the role you want to assign.
- Set the **service** list to the name of the service.
- Set the **action** variable to **member**.

This is the modified Ansible playbook file for the current example:

```
---
- name: Playbook to manage IPA role with members.
  hosts: ipaserver
  become: yes
  gather_facts: no

  tasks:
  - iparole:
    ipaadmin_password: Secret123
    name: web_administrator
    service:
    - HTTP/client01.idm.example.com
    action: member
```

5. Save the file.
6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i ~/<MyPlaybooks>/inventory role-member-service-present-copy.yml
```

Additional resources

- For more information on how to use Ansible Vault to store a password in a separate file or to encrypt it as a variable in the playbook file, see [Encrypting content with Ansible Vault](#).

- For more information on the concept of role in IdM, see [Roles in IdM](#).
- For more sample Ansible playbooks that use the **iparole** module, see the **README-role** Markdown file available in the **/usr/share/doc/ansible-freeipa/** directory. The file also contains the definitions of the **iparole** variables.
- For more sample Ansible playbooks that use the **iparole** module, see the **/usr/share/doc/ansible-freeipa/playbooks/iparole** directory.

11.11. USING ANSIBLE TO ENSURE A HOST IS A MEMBER OF AN IDM RBAC ROLE

As a system administrator managing role-based access control in Identity Management (IdM), you may want to ensure that a specific host or host group is associated with a specific role. The following example describes how to ensure that the custom **web_administrator** role can manage the **client01.idm.example.com** IdM host on which the **HTTP** service is running.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible control node.
- You have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server on which you want to do the configuring.
- Your Ansible inventory file is located in the **~/<MyPlaybooks>/** directory.
- The **web_administrator** role exists in IdM.
- The **client01.idm.example.com** host exists in IdM.

Procedure

1. Navigate to the **~/<MyPlaybooks>/** directory:

```
$ cd ~/<MyPlaybooks>/
```

2. Make a copy of the **role-member-host-present.yml** file located in the **/usr/share/doc/ansible-freeipa/playbooks/role/** directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/role/role-member-host-present.yml role-member-host-present-copy.yml
```

3. Open the **role-member-host-present-copy.yml** Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the **iparole** task section:
 - Set the **ipaadmin_password** variable to the password of the IdM administrator.
 - Set the **name** variable to the name of the role you want to assign.
 - Set the **host** list to the name of the host.

This is the modified Ansible playbook file for the current example:


```

---
- name: Playbook to manage IPA role with members.
  hosts: ipaserver
  become: yes
  gather_facts: no

  tasks:
  - iparole:
    ipaadmin_password: Secret123
    name: web_administrator
    host:
    - client01.idm.example.com
    action: member

```

5. Save the file.
6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i ~/<MyPlaybooks>/inventory role-member-host-present-copy.yml
```

Additional resources

- For more information on how to use Ansible Vault to store a password in a separate file or to encrypt it as a variable in the playbook file, see [Encrypting content with Ansible Vault](#).
- For more information on the concept of role in IdM, see [Roles in IdM](#).
- For more sample Ansible playbooks that use the **iparole** module, see the **README-role** Markdown file available in the **/usr/share/doc/ansible-freeipa/** directory. The file also contains the definitions of the **iparole** variables.
- For more sample Ansible playbooks that use the **iparole** module, see the **/usr/share/doc/ansible-freeipa/playbooks/iparole** directory.

11.12. USING ANSIBLE TO ENSURE A HOST GROUP IS A MEMBER OF AN IDM RBAC ROLE

As a system administrator managing role-based access control in Identity Management (IdM), you may want to ensure that a specific host or host group is associated with a specific role. The following example describes how to ensure that the custom **web_administrator** role can manage the **web_servers** group of IdM hosts on which the **HTTP** service is running.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible control node.
- You have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server on which you want to do the configuring.
- Your Ansible inventory file is located in the **~/<MyPlaybooks>/** directory.

- The **web_administrator** role exists in IdM.
- The **web_servers** host group exists in IdM.

Procedure

1. Navigate to the `~/<MyPlaybooks>/` directory:

```
$ cd ~/<MyPlaybooks>/
```

2. Make a copy of the **role-member-hostgroup-present.yml** file located in the `/usr/share/doc/ansible-freeipa/playbooks/role/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/role/role-member-hostgroup-present.yml role-member-hostgroup-present-copy.yml
```

3. Open the **role-member-hostgroup-present-copy.yml** Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the **iparole** task section:
 - Set the **ipaadmin_password** variable to the password of the IdM administrator.
 - Set the **name** variable to the name of the role you want to assign.
 - Set the **hostgroup** list to the name of the hostgroup.

This is the modified Ansible playbook file for the current example:

```
---
- name: Playbook to manage IPA role with members.
  hosts: ipaserver
  become: yes
  gather_facts: no

  tasks:
  - iparole:
    ipaadmin_password: Secret123
    name: web_administrator
    hostgroup:
    - web_servers
    action: member
```

5. Save the file.
6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i ~/<MyPlaybooks>/inventory role-member-hostgroup-present-copy.yml
```

Additional resources

- For more information on how to use Ansible Vault to store a password in a separate file or to encrypt it as a variable in the playbook file, see [Encrypting content with Ansible Vault](#).

- For more information on the concept of role in IdM, see [Roles in IdM](#).
- For more sample Ansible playbooks that use the **iparole** module, see the **README-role** Markdown file available in the **/usr/share/doc/ansible-freeipa/** directory. The file also contains the definitions of the **iparole** variables.
- For more sample Ansible playbooks that use the **iparole** module, see the **/usr/share/doc/ansible-freeipa/playbooks/iparole** directory.

CHAPTER 12. USING ANSIBLE PLAYBOOKS TO MANAGE RBAC PRIVILEGES

Role-based access control (RBAC) is a policy-neutral access-control mechanism defined around roles, privileges, and permissions. Especially in large companies, using RBAC can help create a hierarchical system of administrators with their individual areas of responsibility.

This chapter describes the following operations for using Ansible playbooks to manage RBAC privileges in Identity Management (IdM):

- [Using Ansible to ensure a custom RBAC privilege is present](#)
- [Using Ansible to ensure member permissions are present in a custom IdM RBAC privilege](#)
- [Using Ansible to ensure an IdM RBAC privilege does not include a permission](#)
- [Using Ansible to rename a custom IdM RBAC privilege](#)
- [Using Ansible to ensure an IdM RBAC privilege is absent](#)

Prerequisites

- You understand the [concepts and principles of RBAC](#).

12.1. USING ANSIBLE TO ENSURE A CUSTOM IDM RBAC PRIVILEGE IS PRESENT

To have a fully-functioning custom privilege in Identity Management (IdM) role-based access control (RBAC), you need to proceed in stages:

1. Create a privilege with no permissions attached.
2. Add permissions of your choice to the privilege.

The following procedure describes how to create an empty privilege using an Ansible playbook so that you can later add permissions to it. The example describes how to create a privilege named **full_host_administration** that is meant to combine all IdM permissions related to host administration.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible control node.
- You have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server on which you want to do the configuring.
- Your Ansible inventory file is located in the `~/MyPlaybooks/` directory.

Procedure

1. Navigate to the `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks/
```

2. Make a copy of the **privilege-present.yml** file located in the `/usr/share/doc/ansible-freeipa/playbooks/privilege/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/privilege/privilege-present.yml privilege-present-copy.yml
```

3. Open the **privilege-present-copy.yml** Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the **ipaprivilege** task section:
 - Set the **ipaadmin_password** variable to the password of the IdM administrator.
 - Set the **name** variable to the name of the new privilege, **full_host_administration**.
 - Optionally, describe the privilege using the **description** variable.

This is the modified Ansible playbook file for the current example:

```
---
- name: Privilege present example
  hosts: ipaserver
  become: true

  tasks:
    - name: Ensure privilege full_host_administration is present
      ipaprivilege:
        ipaadmin_password: Secret123
        name: full_host_administration
        description: This privilege combines all IdM permissions related to host
                    administration
```

5. Save the file.
6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i inventory privilege-present-copy.yml
```

12.2. USING ANSIBLE TO ENSURE MEMBER PERMISSIONS ARE PRESENT IN A CUSTOM IDM RBAC PRIVILEGE

To have a fully-functioning custom privilege in Identity Management (IdM) role-based access control (RBAC), you need to proceed in stages:

1. Create a privilege with no permissions attached.
2. Add permissions of your choice to the privilege.

The following procedure describes how to use an Ansible playbook to add permissions to a privilege created in the previous step. The example describes how to add all IdM permissions related to host administration to a privilege named **full_host_administration**. By default, the permissions are distributed between the **Host Enrollment**, **Host Administrators** and **Host Group Administrator** privileges.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible control node.
- You have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server on which you want to do the configuring.
- Your Ansible inventory file is located in the `~/MyPlaybooks/` directory.
- The **full_host_administration** privilege exists. For information on how to create a privilege using Ansible, see [Using Ansible to ensure a custom IdM RBAC privilege is present](#) .

Procedure

1. Navigate to the `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks/
```

2. Make a copy of the **privilege-member-present.yml** file located in the `/usr/share/doc/ansible-freeipa/playbooks/privilege/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/privilege/privilege-member-present.yml
privilege-member-present-copy.yml
```

3. Open the **privilege-member-present-copy.yml** Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the **ipaprivilege** task section:
 - Adapt the **name** of the task to correspond to your use case.
 - Set the **ipaadmin_password** variable to the password of the IdM administrator.
 - Set the **name** variable to the name of the privilege.
 - Set the **permission** list to the names of the permissions that you want to include in the privilege.
 - Make sure that the **action** variable is set to **member**.

This is the modified Ansible playbook file for the current example:

```
---
- name: Privilege member present example
  hosts: ipaserver
  become: true

  tasks:
    - name: Ensure that permissions are present for the "full_host_administration" privilege
      ipaprivilege:
        ipaadmin_password: Secret123
        name: full_host_administration
        permission:
          - "System: Add krbPrincipalName to a Host"
          - "System: Enroll a Host"
          - "System: Manage Host Certificates"
          - "System: Manage Host Enrollment Password"
```

```

- "System: Manage Host Keytab"
- "System: Manage Host Principals"
- "Retrieve Certificates from the CA"
- "Revoke Certificate"
- "System: Add Hosts"
- "System: Add krbPrincipalName to a Host"
- "System: Enroll a Host"
- "System: Manage Host Certificates"
- "System: Manage Host Enrollment Password"
- "System: Manage Host Keytab"
- "System: Manage Host Keytab Permissions"
- "System: Manage Host Principals"
- "System: Manage Host SSH Public Keys"
- "System: Manage Service Keytab"
- "System: Manage Service Keytab Permissions"
- "System: Modify Hosts"
- "System: Remove Hosts"
- "System: Add Hostgroups"
- "System: Modify Hostgroup Membership"
- "System: Modify Hostgroups"
- "System: Remove Hostgroups"

```

5. Save the file.

6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i inventory privilege-member-present-copy.yml
```

12.3. USING ANSIBLE TO ENSURE AN IDM RBAC PRIVILEGE DOES NOT INCLUDE A PERMISSION

As a system administrator of Identity Management (IdM), you can customize the IdM role-based access control.

The following procedure describes how to use an Ansible playbook to remove a permission from a privilege. The example describes how to remove the **Request Certificates ignoring CA ACLs** permission from the default **Certificate Administrators** privilege because, for example, the administrator considers it a security risk.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible control node.
- You have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server on which you want to do the configuring.
- Your Ansible inventory file is located in the `~/MyPlaybooks/` directory.

Procedure

1. Navigate to the `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks/
```

2. Make a copy of the **privilege-member-present.yml** file located in the **/usr/share/doc/ansible-freeipa/playbooks/privilege/** directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/privilege/privilege-member-absent.yml  
privilege-member-absent-copy.yml
```

3. Open the **privilege-member-absent-copy.yml** Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the **ipaprivilege** task section:
 - Adapt the **name** of the task to correspond to your use case.
 - Set the **ipaadmin_password** variable to the password of the IdM administrator.
 - Set the **name** variable to the name of the privilege.
 - Set the **permission** list to the names of the permissions that you want to remove from the privilege.
 - Make sure that the **action** variable is set to **member**.
 - Make sure that the **state** variable is set to **absent**.

This is the modified Ansible playbook file for the current example:

```
---
- name: Privilege absent example
  hosts: ipaserver
  become: true

  tasks:
    - name: Ensure that the "Request Certificate ignoring CA ACLs" permission is absent from
      the "Certificate Administrators" privilege
      ipaprivilege:
        ipaadmin_password: Secret123
        name: Certificate Administrators
        permission:
          - "Request Certificate ignoring CA ACLs"
        action: member
        state: absent
```

5. Save the file.
6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i inventory privilege-member-absent-copy.yml
```

12.4. USING ANSIBLE TO RENAME A CUSTOM IDM RBAC PRIVILEGE

As a system administrator of Identity Management (IdM), you can customize the IdM role-based access control.

The following procedure describes how to rename a privilege because, for example, you have removed a few permissions from it. As a result, the name of the privilege is no longer accurate. In the example, the administrator renames a **full_host_administration** privilege to **limited_host_administration**.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible control node.
- You have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server on which you want to do the configuring.
- Your Ansible inventory file is located in the `~/MyPlaybooks/` directory.
- The **full_host_administration** privilege exists. For more information on how to add a privilege, see [Using Ansible to ensure a custom IdM RBAC privilege is present](#) .

Procedure

1. Navigate to the `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks/
```

2. Make a copy of the **privilege-present.yml** file located in the `/usr/share/doc/ansible-freeipa/playbooks/privilege/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/privilege/privilege-present.yml rename-privilege.yml
```

3. Open the **rename-privilege.yml** Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the **ipaprivilege** task section:
 - Set the **ipaadmin_password** variable to the password of the IdM administrator.
 - Set the **name** variable to the current name of the privilege.
 - Add the **rename** variable and set it to the new name of the privilege.
 - Add the **state** variable and set it to **renamed**.
5. Rename the playbook itself, for example:

```
---
- name: Rename a privilege
  hosts: ipaserver
  become: true
```

6. Rename the task in the playbook, for example:

```
[...]
tasks:
- name: Ensure the full_host_administration privilege is renamed to
```

limited_host_administration

```
ipaprivilege:
[...]
```

This is the modified Ansible playbook file for the current example:

```
---
- name: Rename a privilege
  hosts: ipaserver
  become: true

  tasks:
    - name: Ensure the full_host_administration privilege is renamed to
      limited_host_administration
      ipaprivilege:
        ipaadmin_password: Secret123
        name: full_host_administration
        rename: limited_host_administration
        state: renamed
```

7. Save the file.
8. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i inventory rename-privilege.yml
```

12.5. USING ANSIBLE TO ENSURE AN IDM RBAC PRIVILEGE IS ABSENT

As a system administrator of Identity Management (IdM), you can customize the IdM role-based access control. The following procedure describes how to use an Ansible playbook to ensure that an RBAC privilege is absent. The example describes how to ensure that the **CA administrator** privilege is absent. As a result of the procedure, the **admin** administrator becomes the only user capable of managing certificate authorities in IdM.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible control node.
- You have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server on which you want to do the configuring.
- Your Ansible inventory file is located in the `~/MyPlaybooks/` directory.

Procedure

1. Navigate to the `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks/
```

2. Make a copy of the **privilege-absent.yml** file located in the `/usr/share/doc/ansible-freeipa/playbooks/privilege/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/privilege/privilege-absent.yml privilege-absent-copy.yml
```

3. Open the **privilege-absent-copy.yml** Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the **ipaprivilege** task section:
 - Set the **ipaadmin_password** variable to the password of the IdM administrator.
 - Set the **name** variable to the name of the privilege you want to remove.
 - Make sure that the **state** variable is set it to **absent**.
5. Rename the task in the playbook, for example:

```
[...]
tasks:
- name: Ensure privilege "CA administrator" is absent
  ipaprivilege:
  [...]
```

This is the modified Ansible playbook file for the current example:

```
---
- name: Privilege absent example
  hosts: ipaserver
  become: true

  tasks:
  - name: Ensure privilege "CA administrator" is absent
    ipaprivilege:
      ipaadmin_password: Secret123
      name: CA administrator
      state: absent
```

6. Save the file.
7. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i inventory privilege-absent-copy.yml
```

Additional resources

- For more information on the concept of a privilege in IdM RBAC, see [Privileges in IdM](#).
- For more information on the concept of a permission in IdM RBAC, see [Permissions in IdM](#).
- For more sample Ansible playbooks that use the **ipaprivilege** module, see the **README-privilege** file available in the **/usr/share/doc/ansible-freeipa/** directory. The file also contains the definitions of the **ipaprivilege** variables.
- For more sample Ansible playbooks that use the **ipaprivilege** module, see the **/usr/share/doc/ansible-freeipa/playbooks/ipaprivilege** directory.

CHAPTER 13. USING ANSIBLE PLAYBOOKS TO MANAGE RBAC PERMISSIONS IN IDM

Role-based access control (RBAC) is a policy-neutral access control mechanism defined around roles, privileges, and permissions. Especially in large companies, using RBAC can help create a hierarchical system of administrators with their individual areas of responsibility.

This chapter describes the following operations performed when managing RBAC permissions in Identity Management (IdM) using Ansible playbooks:

- [Using Ansible to ensure an RBAC permission is present](#)
- [Using Ansible to ensure an RBAC permission with an attribute is present](#)
- [Using Ansible to ensure an RBAC permission is absent](#)
- [Using Ansible to ensure an attribute is a member of an IdM RBAC permission](#)
- [Using Ansible to ensure an attribute is not a member of an IdM RBAC permission](#)
- [Using Ansible to rename an IdM RBAC permission](#)

Prerequisites

- You understand the [concepts and principles of RBAC](#).

13.1. USING ANSIBLE TO ENSURE AN RBAC PERMISSION IS PRESENT

As a system administrator of Identity Management (IdM), you can customize the IdM role-based access control (RBAC).

The following procedure describes how to use an Ansible playbook to ensure a permission is present in IdM so that it can be added to a privilege. The example describes how to ensure the following target state:

- The **MyPermission** permission exists.
- The **MyPermission** permission can only be applied to hosts.
- A user granted a privilege that contains the permission can do all of the following possible operations on an entry:
 - Write
 - Read
 - Search
 - Compare
 - Add
 - Delete

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible control node.
- The example assumes that you have [created and configured](#) the `~/MyPlaybooks/` directory as a central location to store copies of sample playbooks.

Procedure

1. Navigate to the `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks/
```

2. Make a copy of the `permission-present.yml` file located in the `/usr/share/doc/ansible-freeipa/playbooks/permission/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/permission/permission-present.yml
permission-present-copy.yml
```

3. Open the `permission-present-copy.yml` Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the `ipapermission` task section:
 - Adapt the **name** of the task to correspond to your use case.
 - Set the `ipaadmin_password` variable to the password of the IdM administrator.
 - Set the **name** variable to the name of the permission.
 - Set the `object_type` variable to **host**.
 - Set the **right** variable to **all**.

This is the modified Ansible playbook file for the current example:

```
---
- name: Permission present example
  hosts: ipaserver
  become: true

  tasks:
  - name: Ensure that the "MyPermission" permission is present
    ipapermission:
      ipaadmin_password: Secret123
      name: MyPermission
      object_type: host
      right: all
```

5. Save the file.
6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i inventory permission-present-copy.yml
```

13.2. USING ANSIBLE TO ENSURE AN RBAC PERMISSION WITH AN ATTRIBUTE IS PRESENT

As a system administrator of Identity Management (IdM), you can customize the IdM role-based access control (RBAC).

The following procedure describes how to use an Ansible playbook to ensure a permission is present in IdM so that it can be added to a privilege. The example describes how to ensure the following target state:

- The **MyPermission** permission exists.
- The **MyPermission** permission can only be used to add hosts.
- A user granted a privilege that contains the permission can do all of the following possible operations on a host entry:
 - Write
 - Read
 - Search
 - Compare
 - Add
 - Delete
- The host entries created by a user that is granted a privilege that contains the **MyPermission** permission can have a **description** value.



NOTE

The type of attribute that you can specify when creating or modifying a permission is not constrained by the IdM LDAP schema. However, specifying, for example, **attrs: car_licence** if the **object_type** is **host** later results in the **ipa: ERROR: attribute "car-licence" not allowed** error message when you try to exercise the permission and add a specific car licence value to a host.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible control node.
- The example assumes that you have [created and configured](#) the `~/MyPlaybooks/` directory as a central location to store copies of sample playbooks.

Procedure

1. Navigate to the `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks/
```

2. Make a copy of the **permission-present.yml** file located in the **/usr/share/doc/ansible-freeipa/playbooks/permission/** directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/permission/permission-present.yml
permission-present-with-attribute.yml
```

3. Open the **permission-present-with-attribute.yml** Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the **ipapermission** task section:
 - Adapt the **name** of the task to correspond to your use case.
 - Set the **ipaadmin_password** variable to the password of the IdM administrator.
 - Set the **name** variable to the name of the permission.
 - Set the **object_type** variable to **host**.
 - Set the **right** variable to **all**.
 - Set the **attrs** variable to **description**.

This is the modified Ansible playbook file for the current example:

```
---
- name: Permission present example
  hosts: ipaserver
  become: true

  tasks:
  - name: Ensure that the "MyPermission" permission is present with an attribute
    ipapermission:
      ipaadmin_password: Secret123
      name: MyPermission
      object_type: host
      right: all
      attrs: description
```

5. Save the file.
6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i inventory permission-present-with-attribute.yml
```

Additional resources

- For more information on the IdM schema, see [User and group schema](#) in *Linux Domain Identity, Authentication and Policy Guide* in RHEL 7.

13.3. USING ANSIBLE TO ENSURE AN RBAC PERMISSION IS ABSENT

As a system administrator of Identity Management (IdM), you can customize the IdM role-based access control (RBAC).

The following procedure describes how to use an Ansible playbook to ensure a permission is absent in IdM so that it cannot be added to a privilege.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible control node.
- The example assumes that you have [created and configured](#) the `~/MyPlaybooks/` directory as a central location to store copies of sample playbooks.

Procedure

1. Navigate to the `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks/
```

2. Make a copy of the **permission-absent.yml** file located in the `/usr/share/doc/ansible-freeipa/playbooks/permission/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/permission/permission-absent.yml  
permission-absent-copy.yml
```

3. Open the **permission-absent-copy.yml** Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the **ipapermission** task section:
 - Adapt the **name** of the task to correspond to your use case.
 - Set the **ipaadmin_password** variable to the password of the IdM administrator.
 - Set the **name** variable to the name of the permission.

This is the modified Ansible playbook file for the current example:

```
---  
- name: Permission absent example  
  hosts: ipaserver  
  become: true  
  
  tasks:  
    - name: Ensure that the "MyPermission" permission is absent  
      ipapermission:  
        ipaadmin_password: Secret123  
        name: MyPermission  
        state: absent
```

5. Save the file.
6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i inventory permission-absent-copy.yml
```


13.4. USING ANSIBLE TO ENSURE AN ATTRIBUTE IS A MEMBER OF AN IDM RBAC PERMISSION

As a system administrator of Identity Management (IdM), you can customize the IdM role-based access control (RBAC).

The following procedure describes how to use an Ansible playbook to ensure that an attribute is a member of an RBAC permission in IdM. As a result, a user with the permission can create entries that have the attribute.

The example describes how to ensure that the host entries created by a user with a privilege that contains the **MyPermission** permission can have **gecos** and **description** values.



NOTE

The type of attribute that you can specify when creating or modifying a permission is not constrained by the IdM LDAP schema. However, specifying, for example, **attrs: car_licence** if the **object_type** is **host** later results in the **ipa: ERROR: attribute "car-licence" not allowed** error message when you try to exercise the permission and add a specific car licence value to a host.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible control node.
- The example assumes that you have [created and configured](#) the `~/MyPlaybooks/` directory as a central location to store copies of sample playbooks.
- The **MyPermission** permission exists.

Procedure

1. Navigate to the `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks/
```

2. Make a copy of the **permission-member-present.yml** file located in the `/usr/share/doc/ansible-freeipa/playbooks/permission/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/permission/permission-member-present.yml permission-member-present-copy.yml
```

3. Open the **permission-member-present-copy.yml** Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the **ipapermission** task section:
 - Adapt the **name** of the task to correspond to your use case.
 - Set the **ipaadmin_password** variable to the password of the IdM administrator.
 - Set the **name** variable to the name of the permission.
 - Set the **attrs** list to the **description** and **gecos** variables.

- Make sure the **action** variable is set to **member**.

This is the modified Ansible playbook file for the current example:

```
---
- name: Permission member present example
  hosts: ipaserver
  become: true

  tasks:
    - name: Ensure that the "gecos" and "description" attributes are present in
      "MyPermission"
      ipapermission:
        ipaadmin_password: Secret123
        name: MyPermission
        attrs:
          - description
          - gecoss
        action: member
```

5. Save the file.
6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i inventory permission-member-present-copy.yml
```

13.5. USING ANSIBLE TO ENSURE AN ATTRIBUTE IS NOT A MEMBER OF AN IDM RBAC PERMISSION

As a system administrator of Identity Management (IdM), you can customize the IdM role-based access control (RBAC).

The following procedure describes how to use an Ansible playbook to ensure that an attribute is not a member of an RBAC permission in IdM. As a result, when a user with the permission creates an entry in IdM LDAP, that entry cannot have a value associated with the attribute.

The example describes how to ensure the following target state:

- The **MyPermission** permission exists.
- The host entries created by a user with a privilege that contains the **MyPermission** permission cannot have the **description** attribute.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible control node.
- The example assumes that you have [created and configured](#) the `~/MyPlaybooks/` directory as a central location to store copies of sample playbooks.
- The **MyPermission** permission exists.

Procedure

1. Navigate to the `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks/
```

2. Make a copy of the `permission-member-absent.yml` file located in the `/usr/share/doc/ansible-freeipa/playbooks/permission/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/permission/permission-member-absent.yml permission-member-absent-copy.yml
```

3. Open the `permission-member-absent-copy.yml` Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the `ipapermission` task section:
 - Adapt the **name** of the task to correspond to your use case.
 - Set the **ipaadmin_password** variable to the password of the IdM administrator.
 - Set the **name** variable to the name of the permission.
 - Set the **attrs** variable to **description**.
 - Set the **action** variable to **member**.
 - Make sure the **state** variable is set to **absent**

This is the modified Ansible playbook file for the current example:

```
---
- name: Permission absent example
  hosts: ipaserver
  become: true

  tasks:
  - name: Ensure that an attribute is not a member of "MyPermission"
    ipapermission:
      ipaadmin_password: Secret123
      name: MyPermission
      attrs: description
      action: member
      state: absent
```

5. Save the file.
6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i inventory permission-member-absent-copy.yml
```

13.6. USING ANSIBLE TO RENAME AN IDM RBAC PERMISSION

As a system administrator of Identity Management (IdM), you can customize the IdM role-based access control.

The following procedure describes how to use an Ansible playbook to rename a permission. The example describes how to rename **MyPermission** to **MyNewPermission**.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible control node.
- The example assumes that you have [created and configured](#) the `~/MyPlaybooks/` directory as a central location to store copies of sample playbooks.
- The **MyPermission** exists in IdM.
- The **MyNewPermission** does not exist in IdM.

Procedure

1. Navigate to the `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks/
```

2. Make a copy of the **permission-renamed.yml** file located in the `/usr/share/doc/ansible-freeipa/playbooks/permission/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/permission/permission-renamed.yml  
permission-renamed-copy.yml
```

3. Open the **permission-renamed-copy.yml** Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the **ipapermission** task section:
 - Adapt the **name** of the task to correspond to your use case.
 - Set the **ipaadmin_password** variable to the password of the IdM administrator.
 - Set the **name** variable to the name of the permission.

This is the modified Ansible playbook file for the current example:

```
---  
- name: Permission present example  
  hosts: ipaserver  
  become: true  
  
  tasks:  
  - name: Rename the "MyPermission" permission  
    ipapermission:  
      ipaadmin_password: Secret123  
      name: MyPermission  
      rename: MyNewPermission  
      state: renamed
```

5. Save the file.

6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i inventory permission-renamed-copy.yml
```

13.7. ADDITIONAL RESOURCES

- For more information on the concept of a permission in IdM RBAC, see [Permissions in IdM](#).
- For more information on the concept of a privilege in IdM RBAC, see [Privileges in IdM](#).
- For more sample Ansible playbooks that use the **ipapermission** module, see the **README-permission** file available in the **/usr/share/doc/ansible-freeipa/** directory. The file also contains the definitions of the **ipapermission** variables.
- For more sample Ansible playbooks that use the **ipapermission** module, see the **/usr/share/doc/ansible-freeipa/playbooks/ipapermission** directory.

CHAPTER 14. USING ANSIBLE TO MANAGE THE REPLICATION TOPOLOGY IN IDM

You can maintain multiple Identity Management (IdM) servers and let them replicate each other for redundancy purposes to mitigate or prevent server loss. For example, if one server fails, the other servers keep providing services to the domain. You can also recover the lost server by creating a new replica based on one of the remaining servers.

Data stored on an IdM server is replicated based on replication agreements: when two servers have a replication agreement configured, they share their data. The data that is replicated is stored in the topology **suffixes**. When two replicas have a replication agreement between their suffixes, the suffixes form a topology **segment**.

This chapter describes how to use **Red Hat Ansible Engine** to manage IdM replication agreements, topology segments, and topology suffixes. The chapter contains the following sections:

- [Using Ansible to ensure a replication agreement exists in IdM](#)
- [Using Ansible to ensure replication agreements exist between multiple IdM replicas](#)
- [Using Ansible to check if a replication agreement exists between two replicas](#)
- [Using Ansible to verify that a topology suffix exists in IdM](#)
- [Using Ansible to re-initialize an IdM replica](#)
- [Using Ansible to ensure a replication agreement is absent in IdM](#)

14.1. USING ANSIBLE TO ENSURE A REPLICATION AGREEMENT EXISTS IN IDM

Data stored on an Identity Management (IdM) server is replicated based on replication agreements: when two servers have a replication agreement configured, they share their data. Replication agreements are always bilateral: the data is replicated from the first replica to the other one as well as from the other replica to the first one.

This section describes how to use an Ansible playbook to ensure that a replication agreement of the **domain** type exists between **server.idm.example.com** and **replica.idm.example.com**.

Prerequisites

- Ensure that you understand the recommendations for designing your IdM topology listed in [Connecting the replicas in a topology](#).
- You know the IdM **admin** password.
- You have configured an Ansible control node that meets the following requirements:
 - You are using Ansible version 2.8 or later.
 - You have installed the [ansible-freeipa](#) package.
 - In the **~/MyPlaybooks/** directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server where you are configuring these options.

Procedure

1. Navigate to your `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks/
```

2. Copy the **add-topologysegment.yml** Ansible playbook file located in the `/usr/share/doc/ansible-freeipa/playbooks/topology/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/topology/add-topologysegment.yml
add-topologysegment-copy.yml
```

3. Open the **add-topologysegment-copy.yml** file for editing.
4. Adapt the file by setting the following variables in the **ipatopologysegment** task section:
 - Set the **ipaadmin_password** variable to the password of the IdM **admin**.
 - Set the **suffix** variable to either **domain** or **ca**, depending on what type of segment you want to add.
 - Set the **left** variable to the name of the IdM server that you want to be the left node of the replication agreement.
 - Set the **right** variable to the name of the IdM server that you want to be the right node of the replication agreement.
 - Ensure that the **state** variable is set to **present**.

This is the modified Ansible playbook file for the current example:

```
---
- name: Playbook to handle topologysegment
  hosts: ipaserver
  become: true

  tasks:
  - name: Add topology segment
    ipatopologysegment:
      ipaadmin_password: Secret123
      suffix: domain
      left: server.idm.example.com
      right: replica.idm.example.com
      state: present
```

5. Save the file.
6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i inventory add-topologysegment-copy.yml
```

Additional resources

- For more information on the concept of topology agreements, suffixes, and segments, see [Explaining Replication Agreements, Topology Suffixes, and Topology Segments](#) .

- For more sample Ansible playbooks that use the **ipatopologysegment** module, see:
 - The **README-topology.md** file in the `/usr/share/doc/ansible-freeipa/` directory. This file also contains the definitions of the **ipatopologysegment** variables.
 - The `/usr/share/doc/ansible-freeipa/playbooks/topology` directory.

14.2. USING ANSIBLE TO ENSURE REPLICATION AGREEMENTS EXIST BETWEEN MULTIPLE IDM REPLICAS

Data stored on an Identity Management (IdM) server is replicated based on replication agreements: when two servers have a replication agreement configured, they share their data. Replication agreements are always bilateral: the data is replicated from the first replica to the other one as well as from the other replica to the first one.

This section describes how to ensure replication agreements exist between multiple pairs of replicas in IdM.

Prerequisites

- Ensure that you understand the recommendations for designing your IdM topology listed in [Connecting the replicas in a topology](#)
- You know the IdM **admin** password.
- You have configured an Ansible control node that meets the following requirements:
 - You are using Ansible version 2.8 or later.
 - You have installed the [ansible-freeipa](#) package.
 - In the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server where you are configuring these options.

Procedure

1. Navigate to your `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks/
```

2. Copy the **add-topologysegments.yml** Ansible playbook file located in the `/usr/share/doc/ansible-freeipa/playbooks/topology/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/topology/add-topologysegments.yml
add-topologysegments-copy.yml
```

3. Open the **add-topologysegments-copy.yml** file for editing.
4. Adapt the file by setting the following variables in the **vars** section:
 - Set the **ipaadmin_password** variable to the password of the IdM **admin**.
 - For every topology segment, add a line in the **ipatopology_segments** section and set the following variables:

- Set the **suffix** variable to either **domain** or **ca**, depending on what type of segment you want to add.
 - Set the **left** variable to the name of the IdM server that you want to be the left node of the replication agreement.
 - Set the **right** variable to the name of the IdM server that you want to be the right node of the replication agreement.
5. In the **tasks** section of the **add-topologysegments-copy.yml** file, ensure that the **state** variable is set to **present**.

This is the modified Ansible playbook file for the current example:

```
---
- name: Add topology segments
  hosts: ipaserver
  become: true
  gather_facts: false

  vars:
    ipaadmin_password: Secret123
    ipatopology_segments:
      - {suffix: domain, left: replica1.idm.example.com , right: replica2.idm.example.com }
      - {suffix: domain, left: replica2.idm.example.com , right: replica3.idm.example.com }
      - {suffix: domain, left: replica3.idm.example.com , right: replica4.idm.example.com }
      - {suffix: domain+ca, left: replica4.idm.example.com , right: replica1.idm.example.com }

  tasks:
    - name: Add topology segment
      ipatopologysegment:
        ipaadmin_password: "{{ ipaadmin_password }}"
        suffix: "{{ item.suffix }}"
        name: "{{ item.name | default(omit) }}"
        left: "{{ item.left }}"
        right: "{{ item.right }}"
        state: present
        #state: absent
        #state: checked
        #state: reinitialized
        loop: "{{ ipatopology_segments | default([]) }}"
```

6. Save the file.
7. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i inventory add-topologysegments-copy.yml
```

Additional resources

- For more information on the concept of topology agreements, suffixes, and segments, see [Explaining Replication Agreements, Topology Suffixes, and Topology Segments](#) .
- For more sample Ansible playbooks that use the **ipatopologysegment** module, see:

- The **README-topology.md** file in the `/usr/share/doc/ansible-freeipa/` directory. This file also contains the definitions of the **ipatopologysegment** variables.
- The `/usr/share/doc/ansible-freeipa/playbooks/topology` directory.

14.3. USING ANSIBLE TO CHECK IF A REPLICATION AGREEMENT EXISTS BETWEEN TWO REPLICAS

Data stored on an Identity Management (IdM) server is replicated based on replication agreements: when two servers have a replication agreement configured, they share their data. Replication agreements are always bilateral: the data is replicated from the first replica to the other one as well as from the other replica to the first one.

This section describes how to verify that replication agreements exist between multiple pairs of replicas in IdM.

Prerequisites

- Ensure that you understand the recommendations for designing your Identity Management (IdM) topology listed in [Connecting the replicas in a topology](#).
- You know the IdM **admin** password.
- You have configured an Ansible control node that meets the following requirements:
 - You are using Ansible version 2.8 or later.
 - You have installed the [ansible-freeipa](#) package.
 - In the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server where you are configuring these options.

Procedure

1. Navigate to your `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks/
```

2. Copy the **check-topologysegments.yml** Ansible playbook file located in the `/usr/share/doc/ansible-freeipa/playbooks/topology/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/topology/check-topologysegments.yml
check-topologysegments-copy.yml
```

3. Open the **check-topologysegments-copy.yml** file for editing.
4. Adapt the file by setting the following variables in the **vars** section:
 - Set the **ipaadmin_password** variable to the password of the IdM **admin**.
 - For every topology segment, add a line in the **ipatopology_segments** section and set the following variables:
 - Set the **suffix** variable to either **domain** or **ca**, depending on the type of segment you are adding.

- Set the **left** variable to the name of the IdM server that you want to be the left node of the replication agreement.
 - Set the **right** variable to the name of the IdM server that you want to be the right node of the replication agreement.
5. In the **tasks** section of the **check-topologysegments-copy.yml** file, ensure that the **state** variable is set to **present**.

This is the modified Ansible playbook file for the current example:

```
---
- name: Add topology segments
  hosts: ipaserver
  become: true
  gather_facts: false

  vars:
    ipaadmin_password: Secret123
    ipatopology_segments:
      - {suffix: domain, left: replica1.idm.example.com, right: replica2.idm.example.com }
      - {suffix: domain, left: replica2.idm.example.com , right: replica3.idm.example.com }
      - {suffix: domain, left: replica3.idm.example.com , right: replica4.idm.example.com }
      - {suffix: domain+ca, left: replica4.idm.example.com , right:
        replica1.idm.example.com }

  tasks:
    - name: Check topology segment
      ipatopologysegment:
        ipaadmin_password: "{{ ipaadmin_password }}"
        suffix: "{{ item.suffix }}"
        name: "{{ item.name | default(omit) }}"
        left: "{{ item.left }}"
        right: "{{ item.right }}"
        state: checked
      loop: "{{ ipatopology_segments | default([]) }}"
```

6. Save the file.
7. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i inventory check-topologysegments-copy.yml
```

Additional resources

- For more information on the concept of topology agreements, suffixes, and segments, see [Explaining Replication Agreements, Topology Suffixes, and Topology Segments](#) .
- For more sample Ansible playbooks that use the **ipatopologysegment** module, see:
 - The **README-topology.md** file in the **/usr/share/doc/ansible-freeipa/** directory. This file also contains the definitions of the **ipatopologysegment** variables.
 - The **/usr/share/doc/ansible-freeipa/playbooks/topology** directory.

14.4. USING ANSIBLE TO VERIFY THAT A TOPOLOGY SUFFIX EXISTS IN IDM

In the context of replication agreements in Identity Management (IdM), topology suffixes store the data that is replicated. IdM supports two types of topology suffixes: **domain** and **ca**. Each suffix represents a separate back end, a separate replication topology. When a replication agreement is configured, it joins two topology suffixes of the same type on two different servers.

The **domain** suffix contains all domain-related data, such as users, groups, and policies. The **ca** suffix contains data for the Certificate System component. It is only present on servers with a certificate authority (CA) installed.

This section describes how to use an Ansible playbook to ensure that a topology suffix exists in IdM. The example describes how to ensure that the **domain** suffix exists in IdM.

Prerequisites

- You know the IdM **admin** password.
- You have configured an Ansible control node that meets the following requirements:
 - You are using Ansible version 2.8 or later.
 - You have installed the [ansible-freeipa](#) package.
 - In the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server where you are configuring these options.

Procedure

1. Navigate to your `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks/
```

2. Copy the **verify-topologysuffix.yml** Ansible playbook file located in the `/usr/share/doc/ansible-freeipa/playbooks/topology/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/topology/ verify-topologysuffix.yml
verify-topologysuffix-copy.yml
```

3. Open the **verify-topologysuffix-copy.yml** Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the **ipatopologysuffix** section:
 - Set the **ipaadmin_password** variable to the password of the IdM **admin**.
 - Set the **suffix** variable to **domain**. If you are verifying the presence of the **ca** suffix, set the variable to **ca**.
 - Ensure that the **state** variable is set to **verified**. No other option is possible.

This is the modified Ansible playbook file for the current example:

```
---
- name: Playbook to handle topologysuffix
```

```

hosts: ipaserver
become: true

tasks:
- name: Verify topology suffix
  ipatopologysuffix:
    ipaadmin_password: Secret123
    suffix: domain
    state: verified

```

5. Save the file.
6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i inventory verify-topologysuffix-copy.yml
```

Additional resources

- For more information on the concept of topology agreements, suffixes, and segments, see [Explaining Replication Agreements, Topology Suffixes, and Topology Segments](#) .
- For more sample Ansible playbooks that use the **ipatopologysegment** module, see:
 - The **README-topology.md** file in the `/usr/share/doc/ansible-freeipa/` directory. This file also contains the definitions of the **ipatopologysuffix** variables.
 - The `/usr/share/doc/ansible-freeipa/playbooks/topology` directory.

14.5. USING ANSIBLE TO REINITIALIZE AN IDM REPLICA

If a replica has been offline for a long period of time or its database has been corrupted, you can reinitialize it. reinitialization refreshes the replica with an updated set of data. reinitialization can, for example, be used if an authoritative restore from backup is required.



NOTE

In contrast to replication updates, during which replicas only send changed entries to each other, reinitialization refreshes the whole database.

The local host on which you run the command is the reinitialized replica. To specify the replica from which the data is obtained, use the **direction** option.

This section describes how to use an Ansible playbook to reinitialize the **domain** data on **replica.idm.example.com** from **server.idm.example.com**.

Prerequisites

- You know the IdM **admin** password.
- You have configured an Ansible control node that meets the following requirements:
 - You are using Ansible version 2.8 or later.
 - You have installed the [ansible-freeipa](#) package.

- In the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server where you are configuring these options.

Procedure

1. Navigate to your `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks/
```

2. Copy the `reinitialize-topologysegment.yml` Ansible playbook file located in the `/usr/share/doc/ansible-freeipa/playbooks/topology/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/topology/reinitialize-topologysegment.yml reinitialize-topologysegment-copy.yml
```

3. Open the `reinitialize-topologysegment-copy.yml` file for editing.
4. Adapt the file by setting the following variables in the `ipatopologysegment` section:
 - Set the `ipaadmin_password` variable to the password of the IdM `admin`.
 - Set the `suffix` variable to `domain`. If you are reinitializing the `ca` data, set the variable to `ca`.
 - Set the `left` variable to the left node of the replication agreement.
 - Set the `right` variable to the right node of the replication agreement.
 - Set the `direction` variable to the direction of the reinitializing data. The `left-to-right` direction means that data flows from the left node to the right node.
 - Ensure that the `state` variable is set to `reinitialized`.

This is the modified Ansible playbook file for the current example:

```
---
- name: Playbook to handle topologysegment
  hosts: ipaserver
  become: true

  tasks:
    - name: Reinitialize topology segment
      ipatopologysegment:
        ipaadmin_password: Secret123
        suffix: domain
        left: server.idm.example.com
        right: replica.idm.example.com
        direction: left-to-right
        state: reinitialized
```

5. Save the file.
6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i inventory reinitialize-topologysegment-copy.yml
```

Additional resources

- For more information on the concept of topology agreements, suffixes, and segments, see [Explaining Replication Agreements, Topology Suffixes, and Topology Segments](#) .
- For more sample Ansible playbooks that use the **ipatopologysegment** module, see:
 - The **README-topology.md** file in the `/usr/share/doc/ansible-freeipa/` directory. This file also contains the definitions of the **ipatopologysegment** variables.
 - The `/usr/share/doc/ansible-freeipa/playbooks/topology` directory.

14.6. USING ANSIBLE TO ENSURE A REPLICATION AGREEMENT IS ABSENT IN IDM

Data stored on an Identity Management (IdM) server is replicated based on replication agreements: when two servers have a replication agreement configured, they share their data. Replication agreements are always bilateral: the data is replicated from the first replica to the other one as well as from the other replica to the first one.

This section describes how to ensure a replication agreement between two replicas does not exist in IdM. The example describes how to ensure a replication agreement of the **domain** type does not exist between the **replica01.idm.example.com** and **replica02.idm.example.com** IdM servers.

Prerequisites

- Ensure that you understand the recommendations for designing your IdM topology listed in [Connecting the replicas in a topology](#)
- You know the IdM **admin** password.
- You have configured an Ansible control node that meets the following requirements:
 - You are using Ansible version 2.8 or later.
 - You have installed the [ansible-freeipa](#) package.
 - In the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server where you are configuring these options.

Procedure

1. Navigate to your `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks/
```

2. Copy the **delete-topologysegment.yml** Ansible playbook file located in the `/usr/share/doc/ansible-freeipa/playbooks/topology/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/topology/delete-topologysegment.yml
delete-topologysegment-copy.yml
```

3. Open the **delete-topologysegment-copy.yml** file for editing.
4. Adapt the file by setting the following variables in the **ipatopologysegment** task section:

- Set the **ipaadmin_password** variable to the password of the IdM **admin**.
- Set the **suffix** variable to **domain**. Alternatively, if you are ensuring that the **ca** data are not replicated between the left and right nodes, set the variable to **ca**.
- Set the **left** variable to the name of the IdM server that is the left node of the replication agreement.
- Set the **right** variable to the name of the IdM server that is the right node of the replication agreement.
- Ensure that the **state** variable is set to **absent**.

This is the modified Ansible playbook file for the current example:

```
---
- name: Playbook to handle topologysegment
  hosts: ipaserver
  become: true

  tasks:
  - name: Delete topology segment
    ipatopologysegment:
      ipaadmin_password: Secret123
      suffix: domain
      left: replica01.idm.example.com
      right: replica02.idm.example.com:
      state: absent
```

5. Save the file.
6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i inventory delete-topologysegment-copy.yml
```

Additional resources

- For more information on the concept of topology agreements, suffixes, and segments, see [Explaining Replication Agreements, Topology Suffixes, and Topology Segments](#) .
- For more sample Ansible playbooks that use the **ipatopologysegment** module, see:
 - The **README-topology.md** file in the **/usr/share/doc/ansible-freeipa/** directory. This file also contains the definitions of the **ipatopologysegment** variables.
 - The **/usr/share/doc/ansible-freeipa/playbooks/topology** directory.

14.7. ADDITIONAL RESOURCES

- For more information on how to plan the topology of your IdM deployment, see [Planning the replica topology](#).
- For more information on how to install an IdM replica, see [Installing an IdM replica](#) .

CHAPTER 15. MANAGING HOSTS USING ANSIBLE PLAYBOOKS

Ansible is an automation tool used to configure systems, deploy software, and perform rolling updates. Ansible includes support for Identity Management (IdM), and you can use Ansible modules to automate host management.

This chapter describes the following concepts and operations performed when managing hosts and host entries using Ansible playbooks:

- [Hosts in IdM](#)
- [Host enrollment](#)
- [Ensuring the presence of IdM host entries that are only defined by their **FQDNs**](#)
- [Ensuring the presence of IdM host entries with IP addresses](#)
- [Ensuring the presence of multiple IdM host entries with random passwords](#)
- [Ensuring the presence of an IdM host entry with multiple IP addresses](#)
- [Ensuring the absence of IdM host entries](#)

15.1. HOSTS IN IDM

Identity Management (IdM) manages these identities:

- Users
- Services
- Hosts

A host represents a machine. As an IdM identity, a host has an entry in the IdM LDAP, that is the 389 Directory Server instance of the IdM server.

The host entry in IdM LDAP is used to establish relationships between other hosts and even services within the domain. These relationships are part of *delegating* authorization and control to hosts within the domain. Any host can be used in **host-based access control** (HBAC) rules.

IdM domain establishes a commonality between machines, with common identity information, common policies, and shared services. Any machine that belongs to a domain functions as a client of the domain, which means it uses the services that the domain provides. IdM domain provides three main services specifically for machines:

- DNS
- Kerberos
- Certificate management

Hosts in IdM are closely connected with the services running on them:

- Service entries are associated with a host.

- A host stores both the host and the service Kerberos principals.

15.2. HOST ENROLLMENT

This section describes enrolling hosts as IdM clients and what happens during and after the enrollment. The section compares the enrollment of IdM hosts and IdM users. The section also outlines alternative types of authentication available to hosts.

Enrolling a host consists of:

- Creating a host entry in IdM LDAP: possibly using the [ipa host-add command](#) in IdM CLI, or the equivalent [IdM Web UI operation](#).
- Configuring IdM services on the host, for example the System Security Services Daemon (SSSD), Kerberos, and certmonger, and joining the host to the IdM domain.

The two actions can be performed separately or together.

If performed separately, they allow for dividing the two tasks between two users with different levels of privilege. This is useful for bulk deployments.

The **ipa-client-install** command can perform the two actions together. The command creates a host entry in IdM LDAP if that entry does not exist yet, and configures both the Kerberos and SSSD services for the host. The command brings the host within the IdM domain and allows it to identify the IdM server it will connect with. If the host belongs to a DNS zone managed by IdM, **ipa-client-install** adds DNS records for the host too. The command must be run on the client.

15.2.1. User privileges required for host enrollment

The host enrollment operation requires authentication to prevent an unprivileged user from adding unwanted machines to the IdM domain. The privileges required depend on several factors, for example:

- If a host entry is created separately from running **ipa-client-install**
- If a one-time password (OTP) is used for enrollment

User privileges for optionally manually creating a host entry in IdM LDAP

The user privilege required for creating a host entry in IdM LDAP using the **ipa host-add** CLI command or the IdM Web UI is **Host Administrators**. The **Host Administrators** privilege can be obtained through the **IT Specialist** role.

User privileges for joining the client to the IdM domain

Hosts are configured as IdM clients during the execution of the **ipa-client-install** command. The level of credentials required for executing the **ipa-client-install** command depends on which of the following enrolling scenarios you find yourself in:

- The host entry in IdM LDAP does not exist. For this scenario, you need a full administrator's credentials or the **Host Administrators** role. A full administrator is a member of the **admins** group. The **Host Administrators** role provides privileges to add hosts and enroll hosts. For details about this scenario, see [Installing a client using user credentials: interactive installation](#).
- The host entry in IdM LDAP exists. For this scenario, you need a limited administrator's credentials to execute **ipa-client-install** successfully. The limited administrator in this case has the **Enrollment Administrator** role, which provides the **Host Enrollment** privilege. For details, see [Installing a client using user credentials: interactive installation](#).

- The host entry in IdM LDAP exists, and an OTP has been generated for the host by a full or limited administrator. For this scenario, you can install an IdM client as an ordinary user if you run the **ipa-client-install** command with the **--password** option, supplying the correct OTP. For details, see [Installing a client by using a one-time password: Interactive installation](#).

After enrollment, IdM hosts authenticate every new session to be able to access IdM resources. Machine authentication is required for the IdM server to trust the machine and to accept IdM connections from the client software installed on that machine. After authenticating the client, the IdM server can respond to its requests.

15.2.2. Enrollment and authentication of IdM hosts and users: comparison

There are many similarities between users and hosts in IdM. This section describes some of the similarities that can be observed during the enrollment stage as well as those that concern authentication during the deployment stage.

- The enrollment stage ([Table 15.1, “User and host enrollment”](#)):
 - An administrator can create an LDAP entry for both a user and a host before the user or host actually join IdM: for the stage user, the command is **ipa stageuser-add**; for the host, the command is **ipa host-add**.
 - A file containing a *key table* or, abbreviated, *keytab*, a symmetric key resembling to some extent a user password, is created during the execution of the **ipa-client-install** command on the host, resulting in the host joining the IdM realm. Analogically, a user is asked to create a password when they activate their account, thus joining the IdM realm.
 - While the user password is the default authentication method for a user, the keytab is the default authentication method for a host. The keytab is stored in a file on the host.

Table 15.1. User and host enrollment

Action	User	Host
Pre-enrollment	\$ ipa stageuser-add <i>user_name</i> [-password]	\$ ipa host-add <i>host_name</i> [--random]
Activating the account	\$ ipa stageuser-activate <i>user_name</i>	\$ ipa-client install [--password] (must be run on the host itself)

- The deployment stage ([Table 15.2, “User and host session authentication”](#)):
 - When a user starts a new session, the user authenticates using a password; similarly, every time it is switched on, the host authenticates by presenting its keytab file. The System Security Services Daemon (SSSD) manages this process in the background.
 - If the authentication is successful, the user or host obtains a Kerberos ticket granting ticket (TGT).
 - The TGT is then used to obtain specific tickets for specific services.

Table 15.2. User and host session authentication

	User	Host
Default means of authentication	Password	Keytabs
Starting a session (ordinary user)	\$ kinit <i>user_name</i>	<i>[switch on the host]</i>
The result of successful authentication	TGT to be used to obtain access to specific services	TGT to be used to obtain access to specific services

TGTs and other Kerberos tickets are generated as part of the Kerberos services and policies defined by the server. The initial granting of a Kerberos ticket, the renewing of the Kerberos credentials, and even the destroying of the Kerberos session are all handled automatically by the IdM services.

15.2.3. Alternative authentication options for IdM hosts

Apart from keytabs, IdM supports two other types of machine authentication:

- **SSH keys.** The SSH public key for the host is created and uploaded to the host entry. From there, the System Security Services Daemon (SSSD) uses IdM as an identity provider and can work in conjunction with OpenSSH and other services to reference the public keys located centrally in IdM.
- **Machine certificates.** In this case, the machine uses an SSL certificate that is issued by the IdM server's certificate authority and then stored in IdM's Directory Server. The certificate is then sent to the machine to present when it authenticates to the server. On the client, certificates are managed by a service called [certmonger](#).

15.3. ENSURING THE PRESENCE OF AN IDM HOST ENTRY WITH FQDN USING ANSIBLE PLAYBOOKS

This section describes ensuring the presence of host entries in Identity Management (IdM) using Ansible playbooks. The host entries are only defined by their **fully-qualified domain names** (FQDNs).

Specifying the **FQDN** name of the host is enough if at least one of the following conditions applies:

- The IdM server is not configured to manage DNS.
- The host does not have a static IP address or the IP address is not known at the time the host is configured. Adding a host defined only by an **FQDN** essentially creates a placeholder entry in the IdM DNS service. For example, laptops may be preconfigured as IdM clients, but they do not have IP addresses at the time they are configured. When the DNS service dynamically updates its records, the host's current IP address is detected and its DNS record is updated.



NOTE

Without Ansible, host entries are created in IdM using the **ipa host-add** command. The result of adding a host to IdM is the state of the host being present in IdM. Because of the Ansible reliance on idempotence, to add a host to IdM using Ansible, you must create a playbook in which you define the state of the host as present: **state: present**.

Prerequisites

- You know the IdM administrator password.
- The [ansible-freeipa](#) package is installed on the Ansible controller.

Procedure

1. Create an inventory file, for example **inventory.file**, and define **ipaserver** in it:

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

2. Create an Ansible playbook file with the **FQDN** of the host whose presence in IdM you want to ensure. To simplify this step, you can copy and modify the example in the **/usr/share/doc/ansible-freeipa/playbooks/host/add-host.yml** file:

```
---
- name: Host present
  hosts: ipaserver
  become: true

  tasks:
  - name: Host host01.idm.example.com present
    ipahost:
      ipadmin_password: MySecret123
      name: host01.idm.example.com
      state: present
      force: yes
```

3. Run the playbook:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
path_to_playbooks_directory/ensure-host-is-present.yml
```



NOTE

The procedure results in a host entry in the IdM LDAP server being created but not in enrolling the host into the IdM Kerberos realm. For that, you must deploy the host as an IdM client. For details, see [Installing an Identity Management client using an Ansible playbook](#).

Verification steps

1. Log in to your IdM server as admin:

```
$ ssh admin@server.idm.example.com
Password:
```

2. Enter the **ipa host-show** command and specify the name of the host:

```
$ ipa host-show host01.idm.example.com
Host name: host01.idm.example.com
Principal name: host/host01.idm.example.com@IDM.EXAMPLE.COM
Principal alias: host/host01.idm.example.com@IDM.EXAMPLE.COM
Password: False
Keytab: False
Managed by: host01.idm.example.com
```

The output confirms that **host01.idm.example.com** exists in IdM.

15.4. ENSURING THE PRESENCE OF AN IDM HOST ENTRY WITH DNS INFORMATION USING ANSIBLE PLAYBOOKS

This section describes ensuring the presence of host entries in Identity Management (IdM) using Ansible playbooks. The host entries are defined by their **fully-qualified domain names** (FQDNs) and their IP addresses.



NOTE

Without Ansible, host entries are created in IdM using the **ipa host-add** command. The result of adding a host to IdM is the state of the host being present in IdM. Because of the Ansible reliance on idempotence, to add a host to IdM using Ansible, you must create a playbook in which you define the state of the host as present: **state: present**.

Prerequisites

- You know the IdM administrator password.
- The [ansible-freeipa](#) package is installed on the Ansible controller.

Procedure

1. Create an inventory file, for example **inventory.file**, and define **ipaserver** in it:

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

2. Create an Ansible playbook file with the **fully-qualified domain name** (FQDN) of the host whose presence in IdM you want to ensure. In addition, if the IdM server is configured to manage DNS and you know the IP address of the host, specify a value for the **ip_address** parameter. The IP address is necessary for the host to exist in the DNS resource records. To simplify this step, you can copy and modify the example in the **/usr/share/doc/ansible-freeipa/playbooks/host/host-present.yml** file. You can also include other, additional information:

```
---
- name: Host present
  hosts: ipaserver
```

```

become: true

tasks:
- name: Ensure host01.idm.example.com is present
  ipahost:
    ipadmin_password: MySecret123
    name: host01.idm.example.com
    description: Example host
    ip_address: 192.168.0.123
    locality: Lab
    ns_host_location: Lab
    ns_os_version: CentOS 7
    ns_hardware_platform: Lenovo T61
    mac_address:
      - "08:00:27:E3:B1:2D"
      - "52:54:00:BD:97:1E"
    state: present

```

3. Run the playbook:

```

$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
  path_to_playbooks_directory/ensure-host-is-present.yml

```



NOTE

The procedure results in a host entry in the IdM LDAP server being created but not in enrolling the host into the IdM Kerberos realm. For that, you must deploy the host as an IdM client. For details, see [Installing an Identity Management client using an Ansible playbook](#).

Verification steps

1. Log in to your IdM server as admin:

```

$ ssh admin@server.idm.example.com
Password:

```

2. Enter the **ipa host-show** command and specify the name of the host:

```

$ ipa host-show host01.idm.example.com
Host name: host01.idm.example.com
Description: Example host
Locality: Lab
Location: Lab
Platform: Lenovo T61
Operating system: CentOS 7
Principal name: host/host01.idm.example.com@IDM.EXAMPLE.COM
Principal alias: host/host01.idm.example.com@IDM.EXAMPLE.COM
MAC address: 08:00:27:E3:B1:2D, 52:54:00:BD:97:1E
Password: False
Keytab: False
Managed by: host01.idm.example.com

```

The output confirms **host01.idm.example.com** exists in IdM.

15.5. ENSURING THE PRESENCE OF MULTIPLE IDM HOST ENTRIES WITH RANDOM PASSWORDS USING ANSIBLE PLAYBOOKS

The **ipahost** module allows the system administrator to ensure the presence or absence of multiple host entries in IdM using just one Ansible task. This section describes how to ensure the presence of multiple host entries that are only defined by their **fully-qualified domain names** (FQDNs). Running the Ansible playbook generates random passwords for the hosts.



NOTE

Without Ansible, host entries are created in IdM using the **ipa host-add** command. The result of adding a host to IdM is the state of the host being present in IdM. Because of the Ansible reliance on idempotence, to add a host to IdM using Ansible, you must create a playbook in which you define the state of the host as present: **state: present**.

Prerequisites

- You know the IdM administrator password.
- The [ansible-freeipa](#) package is installed on the Ansible controller.

Procedure

1. Create an inventory file, for example **inventory.file**, and define **ipaserver** in it:

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

2. Create an Ansible playbook file with the **fully-qualified domain name** (FQDN) of the hosts whose presence in IdM you want to ensure. To make the Ansible playbook generate a random password for each host even when the host already exists in IdM and **update_password** is limited to **on_create**, add the **random: yes** and **force: yes** options. To simplify this step, you can copy and modify the example from the `/usr/share/doc/ansible-freeipa/README-host.md` Markdown file:

```
---
- name: Ensure hosts with random password
  hosts: ipaserver
  become: true

  tasks:
    - name: Hosts host01.idm.example.com and host02.idm.example.com present with random
      passwords
      ipahost:
        ipadmin_password: MySecret123
        hosts:
          - name: host01.idm.example.com
            random: yes
            force: yes
          - name: host02.idm.example.com
            random: yes
            force: yes
        register: ipahost
```


3. Run the playbook:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
path_to_playbooks_directory/ensure-hosts-are-present.yml
[...]
TASK [Hosts host01.idm.example.com and host02.idm.example.com present with random
passwords]
changed: [r8server.idm.example.com] => {"changed": true, "host":
{"host01.idm.example.com": {"randompassword": "0HoIRvjUdH0Ycbf6uYdWTxH"},
"host02.idm.example.com": {"randompassword": "5VdLgrf3wvojmACdHC3uA3s"}}
```



NOTE

To deploy the hosts as IdM clients using random, one-time passwords (OTPs), see [Authorization options for IdM client enrollment using an Ansible playbook](#) or [Installing a client by using a one-time password: Interactive installation](#).

Verification steps

1. Log in to your IdM server as admin:

```
$ ssh admin@server.idm.example.com
Password:
```

2. Enter the **ipa host-show** command and specify the name of one of the hosts:

```
$ ipa host-show host01.idm.example.com
Host name: host01.idm.example.com
Password: True
Keytab: False
Managed by: host01.idm.example.com
```

The output confirms **host01.idm.example.com** exists in IdM with a random password.

15.6. ENSURING THE PRESENCE OF AN IDM HOST ENTRY WITH MULTIPLE IP ADDRESSES USING ANSIBLE PLAYBOOKS

This section describes how to ensure the presence of a host entry in Identity Management (IdM) using Ansible playbooks. The host entry is defined by its **fully-qualified domain name** (FQDN) and its multiple IP addresses.



NOTE

In contrast to the **ipa host** utility, the Ansible **ipahost** module can ensure the presence or absence of several IPv4 and IPv6 addresses for a host. The **ipa host-mod** command cannot handle IP addresses.

Prerequisites

- You know the IdM administrator password.
- The [ansible-freeipa](#) package is installed on the Ansible controller.

Procedure

1. Create an inventory file, for example **inventory.file**, and define **ipaserver** in it:

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

2. Create an Ansible playbook file. Specify, as the **name** of the **ipahost** variable, the **fully-qualified domain name** (FQDN) of the host whose presence in IdM you want to ensure. Specify each of the multiple IPv4 and IPv6 **ip_address** values on a separate line by using the **- ip_address** syntax. To simplify this step, you can copy and modify the example in the **/usr/share/doc/ansible-freeipa/playbooks/host/host-member-ipaddresses-present.yml** file. You can also include additional information:

```
---
- name: Host member IP addresses present
  hosts: ipaserver
  become: true

  tasks:
  - name: Ensure host101.example.com IP addresses present
    ipahost:
      ipadmin_password: MySecret123
      name: host01.idm.example.com
      ip_address:
        - 192.168.0.123
        - fe80::20c:29ff:fe02:a1b3
        - 192.168.0.124
        - fe80::20c:29ff:fe02:a1b4
      force: yes
```

3. Run the playbook:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
path_to_playbooks_directory/ensure-host-with-multiple-IP-addresses-is-present.yml
```



NOTE

The procedure creates a host entry in the IdM LDAP server but does not enroll the host into the IdM Kerberos realm. For that, you must deploy the host as an IdM client. For details, see [Installing an Identity Management client using an Ansible playbook](#).

Verification steps

1. Log in to your IdM server as admin:

```
$ ssh admin@server.idm.example.com
Password:
```

2. Enter the **ipa host-show** command and specify the name of the host:

```
$ ipa host-show host01.idm.example.com
Principal name: host/host01.idm.example.com@IDM.EXAMPLE.COM
Principal alias: host/host01.idm.example.com@IDM.EXAMPLE.COM
```

```

Password: False
Keytab: False
Managed by: host01.idm.example.com

```

The output confirms that **host01.idm.example.com** exists in IdM.

3. To verify that the multiple IP addresses of the host exist in the IdM DNS records, enter the **ipa dnsrecord-show** command and specify the following information:

- The name of the IdM domain
- The name of the host

```

$ ipa dnsrecord-show idm.example.com host01
[...]
Record name: host01
A record: 192.168.0.123, 192.168.0.124
AAAA record: fe80::20c:29ff:fe02:a1b3, fe80::20c:29ff:fe02:a1b4

```

The output confirms that all the IPv4 and IPv6 addresses specified in the playbook are correctly associated with the **host01.idm.example.com** host entry.

15.7. ENSURING THE ABSENCE OF AN IDM HOST ENTRY USING ANSIBLE PLAYBOOKS

This section describes how to ensure the absence of host entries in Identity Management (IdM) using Ansible playbooks.

Prerequisites

- IdM administrator credentials

Procedure

1. Create an inventory file, for example **inventory.file**, and define **ipaserver** in it:

```

[ipaserver]
server.idm.example.com

```

2. Create an Ansible playbook file with the **fully-qualified domain name** (FQDN) of the host whose absence from IdM you want to ensure. If your IdM domain has integrated DNS, use the **updatedns: yes** option to remove the associated records of any kind for the host from the DNS.

To simplify this step, you can copy and modify the example in the **/usr/share/doc/ansible-freeipa/playbooks/host/delete-host.yml** file:

```

---
- name: Host absent
  hosts: ipaserver
  become: true

  tasks:
    - name: Host host01.idm.example.com absent
      ipahost:

```

```
ipaadmin_password: MySecret123
name: host01.idm.example.com
updatedns: yes
state: absent
```

3. Run the playbook:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
path_to_playbooks_directory/ensure-host-absent.yml
```

NOTE

The procedure results in:

- The host not being present in the IdM Kerberos realm.
- The host entry not being present in the IdM LDAP server.

To remove the specific IdM configuration of system services, such as System Security Services Daemon (SSSD), from the client host itself, you must run the **ipa-client-install --uninstall** command on the client. For details, see [Uninstalling an IdM client](#).

Verification steps

1. Log into **ipaserver** as admin:

```
$ ssh admin@server.idm.example.com
Password:
[admin@server ~]$
```

2. Display information about *host01.idm.example.com*:

```
$ ipa host-show host01.idm.example.com
ipa: ERROR: host01.idm.example.com: host not found
```

The output confirms that the host does not exist in IdM.

Additional resources

- You can see the definitions of the **ipahost** variables as well as sample Ansible playbooks for ensuring the presence, absence, and disablement of hosts in the **/usr/share/doc/ansible-freeipa/README-host.md** Markdown file.
- Additional playbooks are in the **/usr/share/doc/ansible-freeipa/playbooks/host** directory.

CHAPTER 16. MANAGING HOST GROUPS USING ANSIBLE PLAYBOOKS

This chapter introduces [host groups in Identity Management](#) (IdM) and describes using Ansible to perform the following operations involving host groups in Identity Management (IdM):

- [Host groups in IdM](#)
- [Ensuring the presence of IdM host groups](#)
- [Ensuring the presence of hosts in IdM host groups](#)
- [Nesting IdM host groups](#)
- [Ensuring the presence of member managers in IdM host groups](#)
- [Ensuring the absence of hosts from IdM host groups](#)
- [Ensuring the absence of nested host groups from IdM host groups](#)
- [Ensuring the absence of member managers from IdM host groups](#)

16.1. HOST GROUPS IN IDM

IdM host groups can be used to centralize control over important management tasks, particularly access control.

Definition of host groups

A host group is an entity that contains a set of IdM hosts with common access control rules and other characteristics. For example, you can define host groups based on company departments, physical locations, or access control requirements.

A host group in IdM can include:

- IdM servers and clients
- Other IdM host groups

Host groups created by default

By default, the IdM server creates the host group **ipaservers** for all IdM server hosts.

Direct and indirect group members

Group attributes in IdM apply to both direct and indirect members: when host group B is a member of host group A, all members of host group B are considered indirect members of host group A.

16.2. ENSURING THE PRESENCE OF IDM HOST GROUPS USING ANSIBLE PLAYBOOKS

This section describes how to ensure the presence of host groups in Identity Management (IdM) using Ansible playbooks.



NOTE

Without Ansible, host group entries are created in IdM using the **ipa hostgroup-add** command. The result of adding a host group to IdM is the state of the host group being present in IdM. Because of the Ansible reliance on idempotence, to add a host group to IdM using Ansible, you must create a playbook in which you define the state of the host group as present: **state: present**.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible controller.

Procedure

1. Create an inventory file, for example **inventory.file**, and define **ipaserver** in it with the list of IdM servers to target:

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

2. Create an Ansible playbook file with the necessary host group information. For example, to ensure the presence of a host group named **databases**, specify **name: databases** in the **- ipahostgroup** task. To simplify this step, you can copy and modify the example in the **/usr/share/doc/ansible-freeipa/playbooks/user/ensure-hostgroup-is-present.yml** file.

```
---
- name: Playbook to handle hostgroups
  hosts: ipaserver
  become: true

  tasks:
    # Ensure host-group databases is present
    - ipahostgroup:
        ipaadmin_password: MySecret123
        name: databases
        state: present
```

In the playbook, **state: present** signifies a request to add the host group to IdM unless it already exists there.

3. Run the playbook:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
path_to_playbooks_directory/ensure-hostgroup-is-present.yml
```

Verification steps

1. Log into **ipaserver** as admin:

```
$ ssh admin@server.idm.example.com
Password:
[admin@server ~]$
```

2. Request a Kerberos ticket for admin:

```
$ kinit admin
Password for admin@IDM.EXAMPLE.COM:
```

3. Display information about the host group whose presence in IdM you wanted to ensure:

```
$ ipa hostgroup-show databases
Host-group: databases
```

The **databases** host group exists in IdM.

16.3. ENSURING THE PRESENCE OF HOSTS IN IDM HOST GROUPS USING ANSIBLE PLAYBOOKS

This section describes how to ensure the presence of hosts in host groups in Identity Management (IdM) using Ansible playbooks.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible controller.
- The hosts you want to reference in your Ansible playbook exist in IdM. For details, see [Ensuring the presence of an IdM host entry using Ansible playbooks](#).
- The host groups you reference from the Ansible playbook file have been added to IdM. For details, see [Ensuring the presence of IdM host groups using Ansible playbooks](#).

Procedure

1. Create an inventory file, for example **inventory.file**, and define **ipaserver** in it with the list of IdM servers to target:

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

2. Create an Ansible playbook file with the necessary host information. Specify the name of the host group using the **name** parameter of the **ipahostgroup** variable. Specify the name of the host with the **host** parameter of the **ipahostgroup** variable. To simplify this step, you can copy and modify the examples in the **/usr/share/doc/ansible-freeipa/playbooks/hostgroup/ensure-hosts-and-hostgroups-are-present-in-hostgroup.yml** file:

```
---
- name: Playbook to handle hostgroups
  hosts: ipaserver
  become: true

  tasks:
    # Ensure host-group databases is present
    - ipahostgroup:
        ipadmin_password: MySecret123
```

```

name: databases
host:
- db.idm.example.com
action: member

```

This playbook adds the **db.idm.example.com** host to the **databases** host group. The **action: member** line indicates that when the playbook is run, no attempt is made to add the **databases** group itself. Instead, only an attempt is made to add **db.idm.example.com** to **databases**.

3. Run the playbook:

```

$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
path_to_playbooks_directory/ensure-hosts-or-hostgroups-are-present-in-
hostgroup.yml

```

Verification steps

1. Log into **ipaserver** as admin:

```

$ ssh admin@server.idm.example.com
Password:
[admin@server ~]$

```

2. Request a Kerberos ticket for admin:

```

$ kinit admin
Password for admin@IDM.EXAMPLE.COM:

```

3. Display information about a host group to see which hosts are present in it:

```

$ ipa hostgroup-show databases
Host-group: databases
Member hosts: db.idm.example.com

```

The **db.idm.example.com** host is present as a member of the **databases** host group.

16.4. NESTING IDM HOST GROUPS USING ANSIBLE PLAYBOOKS

This section describes ensuring the presence of nested host groups in Identity Management (IdM) host groups using Ansible playbooks.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible controller.
- The host groups you reference from the Ansible playbook file exist in IdM. For details, see [Ensuring the presence of IdM host groups using Ansible playbooks](#) .

Procedure

1. Create an inventory file, for example **inventory.file**, and define **ipaserver** in it with the list of IdM servers to target:

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

2. Create an Ansible playbook file with the necessary host group information. To ensure that a nested host group *A* exists in a host group *B*: in the Ansible playbook, specify, among the - **ipahostgroup** variables, the name of the host group *B* using the **name** variable. Specify the name of the nested hostgroup *A* with the **hostgroup** variable. To simplify this step, you can copy and modify the examples in the **/usr/share/doc/ansible-freeipa/playbooks/hostgroup/ensure-hosts-and-hostgroups-are-present-in-hostgroup.yml** file:

```
---
- name: Playbook to handle hostgroups
  hosts: ipaserver
  become: true

  tasks:
    # Ensure hosts and hostgroups are present in existing databases hostgroup
    - ipahostgroup:
        ipaadmin_password: MySecret123
        name: databases
        hostgroup:
          - mysql-server
          - oracle-server
        action: member
```

This Ansible playbook ensures the presence of the **mysql-server** and **oracle-server** host groups in the **databases** host group. The **action: member** line indicates that when the playbook is run, no attempt is made to add the **databases** group itself to IdM.

3. Run the playbook:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
path_to_playbooks_directory/ensure-hosts-or-hostgroups-are-present-in-
hostgroup.yml
```

Verification steps

1. Log into **ipaserver** as admin:

```
$ ssh admin@server.idm.example.com
Password:
[admin@server /]$
```

2. Request a Kerberos ticket for admin:

```
$ kinit admin
Password for admin@IDM.EXAMPLE.COM:
```

3. Display information about the host group in which nested host groups are present:

\$ ipa hostgroup-show databases

Host-group: databases

Member hosts: db.idm.example.com

Member host-groups: mysql-server, oracle-server

The **mysql-server** and **oracle-server** host groups exist in the **databases** host group.

16.5. ENSURING THE PRESENCE OF MEMBER MANAGERS IN IDM HOST GROUPS USING ANSIBLE PLAYBOOKS

The following procedure describes ensuring the presence of member managers in IdM hosts and host groups using an Ansible playbook.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible controller.
- You must have the name of the host or host group you are adding as member managers and the name of the host group you want them to manage.

Procedure

1. Create an inventory file, for example **inventory.file**, and define **ipaserver** in it:

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

2. Create an Ansible playbook file with the necessary host and host group member management information:

```
---
- name: Playbook to handle host group membership management
  hosts: ipaserver
  become: true

  tasks:
    - name: Ensure member manager user example_member is present for group_name
      ipahostgroup:
        ipadmin_password: MySecret123
        name: group_name
        membermanager_user: example_member

    - name: Ensure member manager group project_admins is present for group_name
      ipahostgroup:
        ipadmin_password: MySecret123
        name: group_name
        membermanager_group: project_admins
```

3. Run the playbook:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
path_to_playbooks_directory/add-member-managers-host-groups.yml
```

Verification steps

You can verify if the **group_name** group contains **example_member** and **project_admins** as member managers by using the **ipa group-show** command:

1. Log into **ipaserver** as administrator:

```
$ ssh admin@server.idm.example.com
Password:
[admin@server /]$
```

2. Display information about *testhostgroup*:

```
ipaserver]$ ipa hostgroup-show group_name
Host-group: group_name
Member hosts: server.idm.example.com
Member host-groups: testhostgroup2
Membership managed by groups: project_admins
Membership managed by users: example_member
```

Additional resources

- See **ipa hostgroup-add-member-manager --help**.
- See the **ipa** man page.

16.6. ENSURING THE ABSENCE OF HOSTS FROM IDM HOST GROUPS USING ANSIBLE PLAYBOOKS

This section describes how to ensure the absence of hosts from host groups in Identity Management (IdM) using Ansible playbooks.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible controller.
- The hosts you want to reference in your Ansible playbook exist in IdM. For details, see [Ensuring the presence of an IdM host entry using Ansible playbooks](#).
- The host groups you reference from the Ansible playbook file exist in IdM. For details, see [Ensuring the presence of IdM host groups using Ansible playbooks](#).

Procedure

1. Create an inventory file, for example **inventory.file**, and define **ipaserver** in it with the list of IdM servers to target:

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

2. Create an Ansible playbook file with the necessary host and host group information. Specify the name of the host group using the **name** parameter of the **ipahostgroup** variable. Specify the name of the host whose absence from the host group you want to ensure using the **host** parameter of the **ipahostgroup** variable. To simplify this step, you can copy and modify the examples in the **/usr/share/doc/ansible-freeipa/playbooks/hostgroup/ensure-hosts-and-hostgroups-are-absent-in-hostgroup.yml** file:

```
---
- name: Playbook to handle hostgroups
  hosts: ipaserver
  become: true

  tasks:
    # Ensure host-group databases is absent
    - ipahostgroup:
        ipaadmin_password: MySecret123
        name: databases
        host:
          - db.idm.example.com
        action: member
        state: absent
```

This playbook ensures the absence of the **db.idm.example.com** host from the **databases** host group. The **action: member** line indicates that when the playbook is run, no attempt is made to remove the **databases** group itself.

3. Run the playbook:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
path_to_playbooks_directory/ensure-hosts-or-hostgroups-are-absent-in-
hostgroup.yml
```

Verification steps

1. Log into **ipaserver** as admin:

```
$ ssh admin@server.idm.example.com
Password:
[admin@server /]$
```

2. Request a Kerberos ticket for admin:

```
$ kinit admin
Password for admin@IDM.EXAMPLE.COM:
```

3. Display information about the host group and the hosts it contains:

```
$ ipa hostgroup-show databases
Host-group: databases
Member host-groups: mysql-server, oracle-server
```

The **db.idm.example.com** host does not exist in the **databases** host group.

16.7. ENSURING THE ABSENCE OF NESTED HOST GROUPS FROM IDM HOST GROUPS USING ANSIBLE PLAYBOOKS

This section describes how to ensure the absence of nested host groups from outer host groups in Identity Management (IdM) using Ansible playbooks.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible controller.
- The host groups you reference from the Ansible playbook file exist in IdM. For details, see [Ensuring the presence of IdM host groups using Ansible playbooks](#) .

Procedure

1. Create an inventory file, for example **inventory.file**, and define **ipaserver** in it with the list of IdM servers to target:

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

2. Create an Ansible playbook file with the necessary host group information. Specify, among the **ipahostgroup** variables, the name of the outer host group using the **name** variable. Specify the name of the nested hostgroup with the **hostgroup** variable. To simplify this step, you can copy and modify the examples in the **/usr/share/doc/ansible-freeipa/playbooks/hostgroup/ensure-hosts-and-hostgroups-are-absent-in-hostgroup.yml** file:

```
---
- name: Playbook to handle hostgroups
  hosts: ipaserver
  become: true

  tasks:
    # Ensure hosts and hostgroups are absent in existing databases hostgroup
    - ipahostgroup:
        ipaadmin_password: MySecret123
        name: databases
        hostgroup:
          - mysql-server
          - oracle-server
        action: member
        state: absent
```

This playbook makes sure that the **mysql-server** and **oracle-server** host groups are absent from the **databases** host group. The **action: member** line indicates that when the playbook is run, no attempt is made to ensure the **databases** group itself is deleted from IdM.

3. Run the playbook:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
path_to_playbooks_directory/ensure-hosts-or-hostgroups-are-absent-in-
hostgroup.yml
```

■

Verification steps

1. Log into **ipaserver** as admin:

```
$ ssh admin@server.idm.example.com
Password:
[admin@server ~]$
```

2. Request a Kerberos ticket for admin:

```
$ kinit admin
Password for admin@IDM.EXAMPLE.COM:
```

3. Display information about the host group from which nested host groups should be absent:

```
$ ipa hostgroup-show databases
Host-group: databases
```

The output confirms that the **mysql-server** and **oracle-server** nested host groups are absent from the outer **databases** host group.

16.8. ENSURING THE ABSENCE OF IDM HOST GROUPS USING ANSIBLE PLAYBOOKS

This section describes how to ensure the absence of host groups in Identity Management (IdM) using Ansible playbooks.



NOTE

Without Ansible, host group entries are removed from IdM using the **ipa hostgroup-del** command. The result of removing a host group from IdM is the state of the host group being absent from IdM. Because of the Ansible reliance on idempotence, to remove a host group from IdM using Ansible, you must create a playbook in which you define the state of the host group as absent: **state: absent**

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible controller.

Procedure

1. Create an inventory file, for example **inventory.file**, and define **ipaserver** in it with the list of IdM servers to target:

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

2. Create an Ansible playbook file with the necessary host group information. To simplify this step, you can copy and modify the example in the **/usr/share/doc/ansible-freeipa/playbooks/user/ensure-hostgroup-is-absent.yml** file.

```
---
- name: Playbook to handle hostgroups
  hosts: ipaserver
  become: true

  tasks:
  - Ensure host-group databases is absent
    ipahostgroup:
      ipaadmin_password: MySecret123
      name: databases
      state: absent
```

This playbook ensures the absence of the **databases** host group from IdM. The **state: absent** means a request to delete the host group from IdM unless it is already deleted.

3. Run the playbook:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
path_to_playbooks_directory/ensure-hostgroup-is-absent.yml
```

Verification steps

1. Log into **ipaserver** as admin:

```
$ ssh admin@server.idm.example.com
Password:
[admin@server ~]$
```

2. Request a Kerberos ticket for admin:

```
$ kinit admin
Password for admin@IDM.EXAMPLE.COM:
```

3. Display information about the host group whose absence you ensured:

```
$ ipa hostgroup-show databases
ipa: ERROR: databases: host group not found
```

The **databases** host group does not exist in IdM.

16.9. ENSURING THE ABSENCE OF MEMBER MANAGERS FROM IDM HOST GROUPS USING ANSIBLE PLAYBOOKS

The following procedure describes ensuring the absence of member managers in IdM hosts and host groups using an Ansible playbook.

Prerequisites

- You know the IdM administrator password.

- You have installed the [ansible-freeipa](#) package on the Ansible controller.
- You must have the name of the user or user group you are removing as member managers and the name of the host group they are managing.

Procedure

1. Create an inventory file, for example **inventory.file**, and define **ipaserver** in it:

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

2. Create an Ansible playbook file with the necessary host and host group member management information:

```
---

- name: Playbook to handle host group membership management
  hosts: ipaserver
  become: true

  tasks:
    - name: Ensure member manager host and host group members are absent for
      group_name
      ipahostgroup:
        ipadmin_password: MySecret123
        name: group_name
        membermanager_user: example_member
        membermanager_group: project_admins
        action: member
        state: absent
```

3. Run the playbook:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
path_to_playbooks_directory/ensure-member-managers-host-groups-are-absent.yml
```

Verification steps

You can verify if the **group_name** group does not contain **example_member** or **project_admins** as member managers by using the **ipa group-show** command:

1. Log into **ipaserver** as administrator:

```
$ ssh admin@server.idm.example.com
Password:
[admin@server /]$
```

2. Display information about *testhostgroup*:

```
ipaserver]$ ipa hostgroup-show group_name
Host-group: group_name
Member hosts: server.idm.example.com
Member host-groups: testhostgroup2
```


Additional resources

- See **ipa hostgroup-add-member-manager --help**.
- See the **ipa** man page.

CHAPTER 17. DEFINING IDM PASSWORD POLICIES

This chapter describes Identity Management (IdM) password policies and how to add a new password policy in IdM using an Ansible playbook.

17.1. WHAT IS A PASSWORD POLICY

A password policy is a set of rules that passwords must meet. For example, a password policy can define the minimum password length and the maximum password lifetime. All users affected by this policy are required to set a sufficiently long password and change it frequently enough to meet the specified conditions. In this way, password policies help reduce the risk of someone discovering and misusing a user's password.

17.2. PASSWORD POLICIES IN IDM

Passwords are the most common way for Identity Management (IdM) users to authenticate to the IdM Kerberos domain. Password policies define the requirements that these IdM user passwords must meet.



NOTE

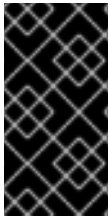
The IdM password policy is set in the underlying LDAP directory, but the Kerberos Key Distribution Center (KDC) enforces the password policy.

[Password policy attributes](#) lists the attributes you can use to define a password policy in IdM.

Table 17.1. Password Policy Attributes

Attribute	Explanation	Example
Max lifetime	The maximum amount of time in days that a password is valid before a user must reset it.	Max lifetime = 90 User passwords are valid only for 90 days. After that, IdM prompts users to change them.
Min lifetime	The minimum amount of time in hours that must pass between two password change operations.	Min lifetime = 1 After users change their passwords, they must wait at least 1 hour before changing them again.
History size	The number of previous passwords that are stored. A user cannot reuse a password from their password history but can reuse old passwords that are not stored.	History size = 0 In this case, the password history is empty and users can reuse any of their previous passwords.

Attribute	Explanation	Example
Character classes	<p>The number of different character classes the user must use in the password. The character classes are:</p> <ul style="list-style-type: none"> * Uppercase characters * Lowercase characters * Digits * Special characters, such as comma (,), period (.), asterisk (*) * Other UTF-8 characters <p>Using a character three or more times in a row decreases the character class by one. For example:</p> <ul style="list-style-type: none"> * Secret1 has 3 character classes: uppercase, lowercase, digits * Secret111 has 2 character classes: uppercase, lowercase, digits, and a -1 penalty for using 1 repeatedly 	<p>Character classes = 0</p> <p>The default number of classes required is 0. To configure the number, run the ipa pwpolicy-mod command with the --minclasses option.</p> <p>See also the Important note below this table.</p>
Min length	<p>The minimum number of characters in a password.</p> <p>If any of the additional password policy options are set, then the minimum length of passwords is 6 regardless of the value to which the Min length option is set.</p>	<p>Min length = 8</p> <p>Users cannot use passwords shorter than 8 characters.</p>
Max failures	<p>The maximum number of failed login attempts before IdM locks the user account.</p>	<p>Max failures = 6</p> <p>IdM locks the user account when the user enters a wrong password 7 times in a row.</p>
Failure reset interval	<p>The amount of time in seconds after which IdM resets the current number of failed login attempts.</p>	<p>Failure reset interval = 60</p> <p>If the user waits for more than 1 minute after the number of failed login attempts defined in Max failures, the user can attempt to log in again without risking a user account lock.</p>
Lockout duration	<p>The amount of time in seconds that the user account is locked after the number of failed login attempts defined in Max failures.</p>	<p>Lockout duration = 600</p> <p>Users with locked accounts are unable to log in for 10 minutes.</p>

**IMPORTANT**

Use the English alphabet and common symbols for the character classes requirement if you have a diverse set of hardware that may not have access to international characters and symbols. For more information about character class policies in passwords, see [What characters are valid in a password?](#) in Red Hat Knowledgebase.

17.3. ENSURING THE PRESENCE OF A PASSWORD POLICY IN IDM USING AN ANSIBLE PLAYBOOK

This section describes how to ensure the presence of a password policy in Identity Management (IdM) using an Ansible playbook.

In the default **global_policy** password policy in IdM, the number of different character classes in the password is set to 0. The history size is also set to 0.

Complete this procedure to enforce a stronger password policy for an IdM group using an Ansible playbook.

**NOTE**

You can only define a password policy for an IdM group. You cannot define a password policy for an individual user.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller.
- You know the IdM administrator password.
- The group for which you are ensuring the presence of a password policy exists in IdM.

Procedure

1. Create an inventory file, for example **inventory.file**, and define the **FQDN** of your IdM server in the **[ipaserver]** section:

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

2. Create your Ansible playbook file that defines the password policy whose presence you want to ensure. To simplify this step, copy and modify the example in the **/usr/share/doc/ansible-freeipa/playbooks/pwpolicy/pwpolicy_present.yml** file:

```
---
- name: Tests
  hosts: ipaserver
  become: true

  tasks:
    - name: Ensure presence of pwpolicy for group ops
      ipapwpolicy:
        ipadmin_password: MySecret123
        name: ops
```

```

minlife: 7
maxlife: 49
history: 5
priority: 1
lockouttime: 300
minlength: 8
minclasses: 4
maxfail: 3
failinterval: 5

```

For details on what the individual variables mean, see [Password policy attributes](#).

3. Run the playbook:

```

$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
  path_to_playbooks_directory/new_pwpolicy_present.yml

```

You have successfully used an Ansible playbook to ensure that a password policy for the **ops** group is present in IdM.



IMPORTANT

The priority of the **ops** password policy is set to *1*, whereas the **global_policy** password policy has no priority set. For this reason, the **ops** policy automatically supersedes **global_policy** for the **ops** group and is enforced immediately.

global_policy serves as a fallback policy when no group policy is set for a user, and it can never take precedence over a group policy.

Additional resources

- For more details about using Ansible to define password policies in IdM and about playbook variables, see the README-pwpolicy.md Markdown file available in the **/usr/share/doc/ansible-freeipa/** directory.
- For more details about how password policy priorities work in IdM, see [Password policy priorities](#) in RHEL 7 documentation.

17.4. ADDITIONAL PASSWORD POLICY OPTIONS IN IDM

As an Identity Management (IdM) administrator, you can strengthen the default password requirements by enabling additional password policy options based on the **libpwquality** feature set. The additional password policy options include the following:

The **--maxrepeat** option

Specifies the maximum acceptable number of same consecutive characters in the new password.

The **--maxsequence** option

Specifies the maximum length of monotonic character sequences in the new password. Examples of such a sequence are **12345** or **fedcb**. Most such passwords will not pass the simplicity check. The only exception is when the sequence is only a minor part of the password.

The **--dictcheck** option

If nonzero, checks whether the password, with possible modifications, matches a word in a dictionary. Currently **libpwquality** performs the dictionary check using the **cracklib** library.

The `--usercheck` option

If nonzero, checks whether the password, with possible modifications, contains the user name in some form. It is not performed for user names shorter than 3 characters.

You cannot apply the additional password policy options to existing passwords. If you apply any of the additional options, IdM automatically sets the `--minlength` option, the minimum number of characters in a password, to **6** characters.



NOTE

In a mixed environment with RHEL 7 and RHEL 8 servers, you can enforce the additional password policy settings only on servers running on RHEL 8.4 and later.

Additional resources:

- [Applying additional password policies to an IdM group](#)
- `pwquality(3)` man page

17.5. APPLYING ADDITIONAL PASSWORD POLICY OPTIONS TO AN IDM GROUP

This section describes how to apply additional password policy options in Identity Management (IdM). The example describes how to strengthen the password policy for the **managers** group by making sure that the new passwords do not contain the users' respective user names and that the passwords contain no more than two identical characters in succession.

Prerequisites

- You are logged in as an IdM administrator.
- The **managers** group exists in IdM.
- The **managers** password policy exists in IdM.

Procedure

1. Apply the user name check to all new passwords suggested by the users in the **managers** group:

```
$ ipa pwpolicy-mod --usercheck=True managers
```



NOTE

If you do not specify the name of the password policy, the default **global_policy** is modified.

2. Set the maximum number of identical consecutive characters to 2 in the **managers** password policy:

```
$ ipa pwpolicy-mod --maxrepeat=2 managers
```

A password now will not be accepted if it contains more than 2 identical consecutive characters. For example, the **eR873mUi111YJQ** combination is unacceptable because it contains three **1s** in succession.

Verification

1. Add a test user named **test_user**:

```
$ ipa user-add test_user
First name: test
Last name: user
-----
Added user "test_user"
-----
```

2. Add the test user to the **managers** group:
 - a. In the IdM Web UI, click **Identity** → **Groups** → **User Groups**.
 - b. Click **managers**.
 - c. Click **Add**.
 - d. In the **Add users into user group 'managers'** page, check **test_user**.
 - e. Click the **>** arrow to move the user to the **Prospective** column.
 - f. Click **Add**.
3. Reset the password for the test user:
 - a. Go to **Identity** → **Users**.
 - b. Click **test_user**.
 - c. In the **Actions** menu, click **Reset Password**.
 - d. Enter a temporary password for the user.
4. On the command line, try to obtain a Kerberos ticket-granting ticket (TGT) for the **test_user**:

```
$ kinit test_user
```

- a. Enter the temporary password.
- b. The system informs you that you must change your password. Enter a password that contains the user name of **test_user**:

```
Password expired. You must change it now.
Enter new password:
Enter it again:
```



NOTE

Kerberos does not have fine-grained error password policy reporting and, in certain cases, does not provide a clear reason why a password was rejected.

- c. The system informs you that the entered password was rejected. Enter a password that contains three or more identical characters in succession:

```
Password change rejected: Password not changed.  
Unspecified password quality failure while trying to change password.  
Please try again.
```

```
Enter new password:  
Enter it again:
```

- d. The system informs you that the entered password was rejected. Enter a password that meets the criteria of the **managers** password policy:

```
Password change rejected: Password not changed.  
Unspecified password quality failure while trying to change password.  
Please try again.
```

```
Enter new password:  
Enter it again:
```

5. View the obtained TGT:

```
$ klist  
Ticket cache: KCM:0:33945  
Default principal: test_user@IDM.EXAMPLE.COM  
  
Valid starting    Expires          Service principal  
07/07/2021 12:44:44 07/08/2021 12:44:44  
krbtgt@IDM.EXAMPLE.COM@IDM.EXAMPLE.COM
```

The **managers** password policy now works correctly for users in the **managers** group.

Additional resources

- [Additional password policies in IdM](#)

CHAPTER 18. GRANTING SUDO ACCESS TO AN IDM USER ON AN IDM CLIENT

18.1. SUDO ACCESS ON AN IDM CLIENT

System administrators can grant **sudo** access to allow non-root users to execute administrative commands that are normally reserved for the **root** user. Consequently, when users need to perform an administrative command normally reserved for the **root** user, they precede that command with **sudo**. After entering their password, the command is executed as if they were the **root** user. To execute a **sudo** command as another user or group, such as a database service account, you can configure a *RunAs alias* for a **sudo** rule.

If a Red Hat Enterprise Linux (RHEL) 8 host is enrolled as an Identity Management (IdM) client, you can specify **sudo** rules defining which IdM users can perform which commands on the host in the following ways:

- Locally in the **/etc/sudoers** file
- Centrally in IdM

This section describes creating a **central sudo rule** for an IdM client using the command line interface (CLI) and the IdM Web UI.

In RHEL 8.4 and later, you can also configure password-less authentication for **sudo** using the Generic Security Service Application Programming Interface (GSSAPI), the native way for UNIX-based operating systems to access and authenticate Kerberos services. You can use the **pam_sss_gss.so** Pluggable Authentication Module (PAM) to invoke GSSAPI authentication via the SSSD service, allowing users to authenticate to the **sudo** command with a valid Kerberos ticket.

Additional resources

- For details on creating local **sudo** rules on a RHEL 8 host, see [Managing sudo access](#).

18.2. GRANTING SUDO ACCESS TO AN IDM USER ON AN IDM CLIENT USING THE CLI

In Identity Management (IdM), you can grant **sudo** access for a specific command to an IdM user account on a specific IdM host. First, add a **sudo** command and then create a **sudo** rule for one or more commands.

For example, complete this procedure to create the **idm_user_reboot sudo** rule to grant the **idm_user** account the permission to run the **/usr/sbin/reboot** command on the **idmclient** machine.

Prerequisites

- You are logged in as IdM administrator.
- You have created a user account for **idm_user** in IdM and unlocked the account by creating a password for the user. For details on adding a new IdM user using the CLI, see [Adding users using the command line](#).
- No local **idm_user** account is present on the **idmclient** host. The **idm_user** user is not listed in the local **/etc/passwd** file.

Procedure

1. Retrieve a Kerberos ticket as the IdM **admin**.

```
[root@idmclient ~]# kinit admin
```

2. Add the **/usr/sbin/reboot** command to the IdM database of **sudo** commands:

```
[root@idmclient ~]# ipa sudocmd-add /usr/sbin/reboot
-----
Added Sudo Command "/usr/sbin/reboot"
-----
Sudo Command: /usr/sbin/reboot
```

3. Create a **sudo** rule named **idm_user_reboot**:

```
[root@idmclient ~]# ipa sudorule-add idm_user_reboot
-----
Added Sudo Rule "idm_user_reboot"
-----
Rule name: idm_user_reboot
Enabled: TRUE
```

4. Add the **/usr/sbin/reboot** command to the **idm_user_reboot** rule:

```
[root@idmclient ~]# ipa sudorule-add-allow-command idm_user_reboot --sudocmds
'/usr/sbin/reboot'
Rule name: idm_user_reboot
Enabled: TRUE
Sudo Allow Commands: /usr/sbin/reboot
-----
Number of members added 1
-----
```

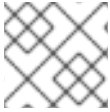
5. Apply the **idm_user_reboot** rule to the IdM **idmclient** host:

```
[root@idmclient ~]# ipa sudorule-add-host idm_user_reboot --hosts
idmclient.idm.example.com
Rule name: idm_user_reboot
Enabled: TRUE
Hosts: idmclient.idm.example.com
Sudo Allow Commands: /usr/sbin/reboot
-----
Number of members added 1
-----
```

6. Add the **idm_user** account to the **idm_user_reboot** rule:

```
[root@idmclient ~]# ipa sudorule-add-user idm_user_reboot --users idm_user
Rule name: idm_user_reboot
Enabled: TRUE
Users: idm_user
Hosts: idmclient.idm.example.com
Sudo Allow Commands: /usr/sbin/reboot
```

 Number of members added 1



NOTE

Propagating the changes from the server to the client can take a few minutes.

Verification steps

1. Log in to the **idmclient** host as the **idm_user** account.
2. Display which **sudo** rules the **idm_user** account is allowed to perform.

```
[idm_user@idmclient ~]$ sudo -l
Matching Defaults entries for idm_user on idmclient:
    !visiblepw, always_set_home, match_group_by_gid, always_query_group_plugin,
    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
    KRB5CCNAME",
    secure_path="/sbin:/bin:/usr/sbin:/usr/bin
```

User **idm_user** may run the following commands on **idmclient**:
(root) /usr/sbin/reboot

3. Reboot the machine using **sudo**. Enter the password for **idm_user** when prompted:

```
[idm_user@idmclient ~]$ sudo /usr/sbin/reboot
[sudo] password for idm_user:
```

18.3. GRANTING SUDO ACCESS TO AN IDM USER ON AN IDM CLIENT USING THE IDM WEB UI

In Identity Management (IdM), you can grant **sudo** access for a specific command to an IdM user account on a specific IdM host. First, add a **sudo** command and then create a **sudo** rule for one or more commands.

Complete this procedure to create the **idm_user_reboot** sudo rule to grant the **idm_user** account the permission to run the **/usr/sbin/reboot** command on the **idmclient** machine.

Prerequisites

- You are logged in as IdM administrator.
- You have created a user account for **idm_user** in IdM and unlocked the account by creating a password for the user. For details on adding a new IdM user using the command-line interface, see [Adding users using the command line](#).

- No local **idm_user** account is present on the **idmclient** host. The **idm_user** user is not listed in the local **/etc/passwd** file.

Procedure

1. Add the **/usr/sbin/reboot** command to the IdM database of **sudo** commands:
 - a. Navigate to **Policy → Sudo → Sudo Commands**.
 - b. Click **Add** in the upper right corner to open the **Add sudo command** dialog box.
 - c. Enter the command you want the user to be able to perform using **sudo: /usr/sbin/reboot**.

Figure 18.1. Adding IdM sudo command

Add sudo command

Sudo Command *

Description

* Required field

Add Add and Add Another Add and Edit Cancel

- d. Click **Add**.
2. Use the new **sudo** command entry to create a sudo rule to allow **idm_user** to reboot the **idmclient** machine:
 - a. Navigate to **Policy → Sudo → Sudo rules**.
 - b. Click **Add** in the upper right corner to open the **Add sudo rule** dialog box.
 - c. Enter the name of the **sudo** rule: **idm_user_reboot**.
 - d. Click **Add and Edit**
 - e. Specify the user:
 - i. In the **Who** section, check the **Specified Users and Groups** radio button.
 - ii. In the **User category the rule applies to** subsection, click **Add** to open the **Add users into sudo rule "idm_user_reboot"** dialog box.
 - iii. In the **Add users into sudo rule "idm_user_reboot"** dialog box in the **Available** column, check the **idm_user** checkbox, and move it to the **Prospective** column.

- iv. Click **Add**.
- f. Specify the host:
 - i. In the **Access this host** section, check the **Specified Hosts and Groups** radio button.
 - ii. In the **Host category this rule applies to** subsection, click **Add** to open the **Add hosts into sudo rule "idm_user_reboot"** dialog box.
 - iii. In the **Add hosts into sudo rule "idm_user_reboot"** dialog box in the **Available** column, check the **idmclient.idm.example.com** checkbox, and move it to the **Prospective** column.
 - iv. Click **Add**.
- g. Specify the commands:
 - i. In the **Command category the rule applies to** subsection of the **Run Commands** section, check the **Specified Commands and Groups** radio button.
 - ii. In the **Sudo Allow Commands** subsection, click **Add** to open the **Add allow sudo commands into sudo rule "idm_user_reboot"** dialog box.
 - iii. In the **Add allow sudo commands into sudo rule "idm_user_reboot"** dialog box in the **Available** column, check the **/usr/sbin/reboot** checkbox, and move it to the **Prospective** column.
 - iv. Click **Add** to return to the **idm_sudo_reboot** page.

Adding IdM sudo rule

+ image::IdM-sudo-rule-WebUI.png[A screenshot of an overview of the sudo rule that was added. There is a "Who" section with a table of users the rule applies to. There is an "Access this host" section with a table of hosts that the rule applies to. There is a "Run Commands" section with a table of commands that pertain to the rule.]

- a. Click **Save** in the top left corner.

The new rule is enabled by default.



NOTE

Propagating the changes from the server to the client can take a few minutes.

Verification steps

1. Log in to **idmclient** as **idm_user**.
2. Reboot the machine using **sudo**. Enter the password for **idm_user** when prompted:

```
$ sudo /usr/sbin/reboot
[sudo] password for idm_user:
```

If the **sudo** rule is configured correctly, the machine reboots.

18.4. CREATING A SUDO RULE ON THE CLI THAT RUNS A COMMAND AS A SERVICE ACCOUNT ON AN IDM CLIENT

In IdM, you can configure a **sudo** rule with a *RunAs alias* to run a **sudo** command as another user or group. For example, you might have an IdM client that hosts a database application, and you need to run commands as the local service account that corresponds to that application.

Use this example to create a **sudo** rule on the command line called **run_third-party-app_report** to allow the **idm_user** account to run the **/opt/third-party-app/bin/report** command as the **thirdpartyapp** service account on the **idmclient** host.

Prerequisites

- You are logged in as IdM administrator.
- You have created a user account for **idm_user** in IdM and unlocked the account by creating a password for the user. For details on adding a new IdM user using the CLI, see [Adding users using the command line](#).
- No local **idm_user** account is present on the **idmclient** host. The **idm_user** user is not listed in the local **/etc/passwd** file.
- You have a custom application named **third-party-app** installed on the **idmclient** host.
- The **report** command for the **third-party-app** application is installed in the **/opt/third-party-app/bin/report** directory.
- You have created a local service account named **thirdpartyapp** to execute commands for the **third-party-app** application.

Procedure

1. Retrieve a Kerberos ticket as the IdM **admin**.

```
[root@idmclient ~]# kinit admin
```

2. Add the **/opt/third-party-app/bin/report** command to the IdM database of **sudo** commands:

```
[root@idmclient ~]# ipa sudocmd-add /opt/third-party-app/bin/report
-----
Added Sudo Command "/opt/third-party-app/bin/report"
-----
Sudo Command: /opt/third-party-app/bin/report
```

3. Create a **sudo** rule named **run_third-party-app_report**:

```
[root@idmclient ~]# ipa sudorule-add run_third-party-app_report
-----
Added Sudo Rule "run_third-party-app_report"
-----
Rule name: run_third-party-app_report
Enabled: TRUE
```

4. Use the **--users=<user>** option to specify the RunAs user for the **sudo** rule **add-runasuser** command:

```
[root@idmclient ~]# ipa sudorule-add-runasuser run_third-party-app_report --
users=thirdpartyapp
Rule name: run_third-party-app_report
Enabled: TRUE
RunAs External User: thirdpartyapp
-----
Number of members added 1
-----
```

The user (or group specified with the **--groups=*** option) can be external to IdM, such as a local service account or an Active Directory user. Do not add a **%** prefix for group names.

5. Add the **/opt/third-party-app/bin/report** command to the **idm_user_reboot** rule:

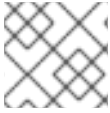
```
[root@idmclient ~]# ipa sudorule-add-allow-command run_third-party-app_report --
sudocmds '/opt/third-party-app/bin/report'
Rule name: run_third-party-app_report
Enabled: TRUE
Sudo Allow Commands: /opt/third-party-app/bin/report
RunAs External User: thirdpartyapp
-----
Number of members added 1
-----
```

6. Apply the **run_third-party-app_report** rule to the IdM **idmclient** host:

```
[root@idmclient ~]# ipa sudorule-add-host run_third-party-app_report --hosts
idmclient.idm.example.com
Rule name: run_third-party-app_report
Enabled: TRUE
Hosts: idmclient.idm.example.com
Sudo Allow Commands: /opt/third-party-app/bin/report
RunAs External User: thirdpartyapp
-----
Number of members added 1
-----
```

7. Add the **idm_user** account to the **run_third-party-app_report** rule:

```
[root@idmclient ~]# ipa sudorule-add-user run_third-party-app_report --users idm_user
Rule name: run_third-party-app_report
Enabled: TRUE
Users: idm_user
Hosts: idmclient.idm.example.com
Sudo Allow Commands: /opt/third-party-app/bin/report
RunAs External User: thirdpartyapp
-----
Number of members added 1
-----
```

**NOTE**

Propagating the changes from the server to the client can take a few minutes.

Verification steps

1. Log in to the **idmclient** host as the **idm_user** account.
2. Test the new sudo rule:
 - a. Display which **sudo** rules the **idm_user** account is allowed to perform.

```
[idm_user@idmclient ~]$ sudo -l
Matching Defaults entries for idm_user@idm.example.com on idmclient:
    !visiblepw, always_set_home, match_group_by_gid, always_query_group_plugin,
    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 KRB5CCNAME",
    secure_path="/sbin\:/bin\:/usr/sbin\:/usr/bin
```

User idm_user@idm.example.com may run the following commands on idmclient:
(thirdpartyapp) /opt/third-party-app/bin/report

- b. Run the **report** command as the **thirdpartyapp** service account.

```
[idm_user@idmclient ~]$ sudo -u thirdpartyapp /opt/third-party-app/bin/report
[sudo] password for idm_user@idm.example.com:
Executing report...
Report successful.
```

18.5. CREATING A SUDO RULE IN THE IDM WEBUI THAT RUNS A COMMAND AS A SERVICE ACCOUNT ON AN IDM CLIENT

In IdM, you can configure a **sudo** rule with a *RunAs alias* to run a **sudo** command as another user or group. For example, you might have an IdM client that hosts a database application, and you need to run commands as the local service account that corresponds to that application.

Use this example to create a **sudo** rule in the IdM WebUI called **run_third-party-app_report** to allow the **idm_user** account to run the **/opt/third-party-app/bin/report** command as the **thirdpartyapp** service account on the **idmclient** host.

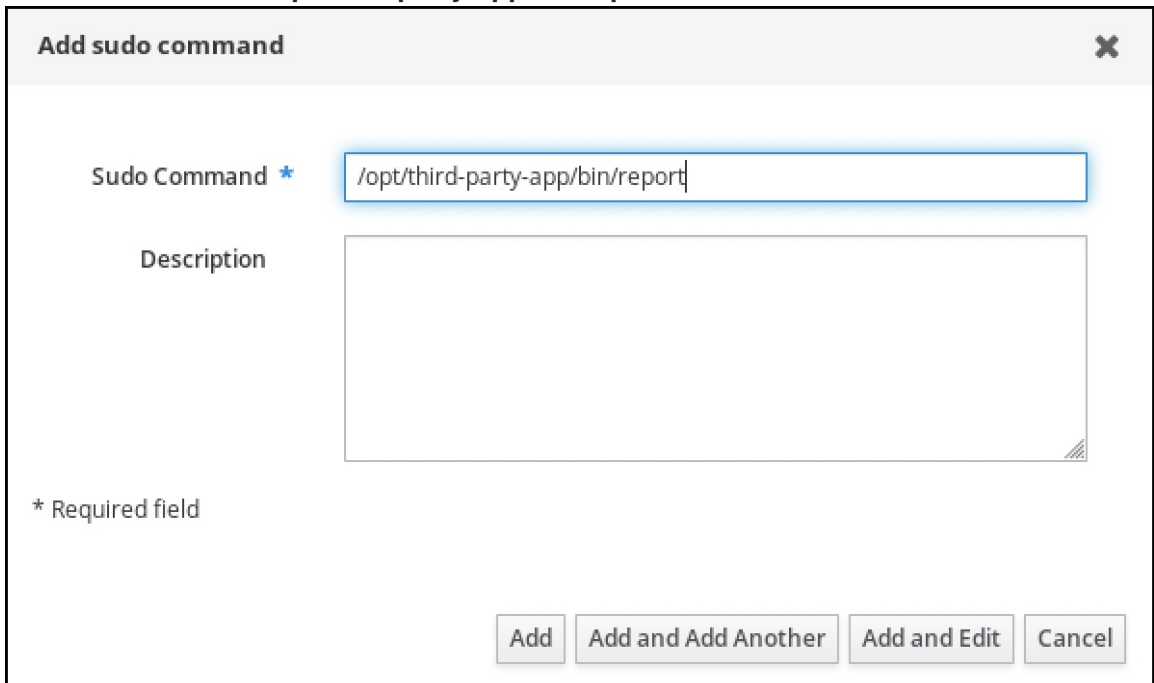
Prerequisites

- You are logged in as IdM administrator.
- You have created a user account for **idm_user** in IdM and unlocked the account by creating a password for the user. For details on adding a new IdM user using the CLI, see [Adding users using the command line](#).

- No local **idm_user** account is present on the **idmclient** host. The **idm_user** user is not listed in the local **/etc/passwd** file.
- You have a custom application named **third-party-app** installed on the **idmclient** host.
- The **report** command for the **third-party-app** application is installed in the **/opt/third-party-app/bin/report** directory.
- You have created a local service account named **thirdpartyapp** to execute commands for the **third-party-app** application.

Procedure

1. Add the **/opt/third-party-app/bin/report** command to the IdM database of **sudo** commands:
 - a. Navigate to **Policy → Sudo → Sudo Commands**.
 - b. Click **Add** in the upper right corner to open the **Add sudo command** dialog box.
 - c. Enter the command: **/opt/third-party-app/bin/report**.



Add sudo command

Sudo Command *

Description

* Required field

Add Add and Add Another Add and Edit Cancel

- d. Click **Add**.
2. Use the new **sudo** command entry to create the new **sudo** rule:
 - a. Navigate to **Policy → Sudo → Sudo rules**.
 - b. Click **Add** in the upper right corner to open the **Add sudo rule** dialog box.
 - c. Enter the name of the **sudo** rule: **run_third-party-app_report**.

Add sudo rule ✕

Rule name *

* Required field

d. Click **Add and Edit**

e. Specify the user:

- i. In the **Who** section, check the **Specified Users and Groups** radio button.
- ii. In the **User category the rule applies to** subsection, click **Add** to open the **Add users into sudo rule "run_third-party-app_report"** dialog box.
- iii. In the **Add users into sudo rule "run_third-party-app_report"** dialog box in the **Available** column, check the **idm_user** checkbox, and move it to the **Prospective** column.

Add users into sudo rule 'run_third-party-app_report' ✕

Available			Prospective	
<input type="checkbox"/>	Users	<div style="border: 2px solid red; padding: 2px; display: inline-block;">></div> <	<input type="checkbox"/>	Users
<input type="checkbox"/>	admin		<input checked="" type="checkbox"/>	idmuser

External

iv. Click **Add**.

f. Specify the host:

- i. In the **Access this host** section, check the **Specified Hosts and Groups** radio button.
- ii. In the **Host category this rule applies to** subsection, click **Add** to open the **Add hosts into sudo rule "run_third-party-app_report"** dialog box.
- iii. In the **Add hosts into sudo rule "run_third-party-app_report"** dialog box in the **Available** column, check the **idmclient.idm.example.com** checkbox, and move it to the **Prospective** column.

Add hosts into sudo rule 'run_third-party-app_report'

Filter available Hosts Filter

Available

<input type="checkbox"/>	Hosts
<input type="checkbox"/>	client2.idm.example.com

Prospective

<input type="checkbox"/>	Hosts
<input checked="" type="checkbox"/>	idmclient.idm.example.com

External

Add Cancel

iv. Click **Add**.

g. Specify the commands:

- i. In the **Command category** the rule applies to subsection of the **Run Commands** section, check the **Specified Commands and Groups** radio button.
- ii. In the **Sudo Allow Commands** subsection, click **Add** to open the **Add allow sudo commands into sudo rule "run_third-party-app_report"** dialog box.
- iii. In the **Add allow sudo commands into sudo rule "run_third-party-app_report"** dialog box in the **Available** column, check the **/opt/third-party-app/bin/report** checkbox, and move it to the **Prospective** column.

Add allow sudo commands into sudo rule 'run_third-party-app_report'

Filter available Sudo Commands Filter

Available

<input type="checkbox"/>	Sudo Commands
<input type="checkbox"/>	help

Prospective

<input type="checkbox"/>	Sudo Commands
<input checked="" type="checkbox"/>	/opt/third-party-app/bin/report

Add Cancel

iv. Click **Add** to return to the **run_third-party-app_report** page.

h. Specify the RunAs user:

- i. In the **As Whom** section, check the **Specified Users and Groups** radio button.
- ii. In the **RunAs Users** subsection, click **Add** to open the **Add RunAs users into sudo rule "run_third-party-app_report"** dialog box.

- iii. In the **Add RunAs users into sudo rule "run_third-party-app_report"** dialog box, enter the **thirdpartyapp** service account in the **External** box and move it to the **Prospective** column.

Add RunAs users into sudo rule 'run_third-party-app_report'

Filter available Users Filter

Available

<input type="checkbox"/>	Users
<input type="checkbox"/>	admin
<input type="checkbox"/>	employee
<input type="checkbox"/>	helpdesk
<input type="checkbox"/>	manager

Prospective

<input type="checkbox"/>	Users
--------------------------	-------

External

thirdpartyapp

> <

Add Cancel

- iv. Click **Add** to return to the **run_third-party-app_report** page.

- i. Click **Save** in the top left corner.

The new rule is enabled by default.

Figure 18.2. Details of the sudo rule

Who

User category the rule applies to: ☐ Anyone ☒ Specified Users and Groups

<input type="checkbox"/> Users	External	Delete + Add
<input type="checkbox"/> idm_user		

☐ User Groups [Delete](#) [+ Add](#)

Access this host

Host category the rule applies to: ☐ Any Host ☒ Specified Hosts and Groups

<input type="checkbox"/> Hosts	External	Delete + Add
<input type="checkbox"/> idmclient.idm.example.com		

☐ Host Groups [Delete](#) [+ Add](#)

Run Commands

Command category the rule applies to: ☐ Any Command ☒ Specified Commands and Groups

Allow

<input type="checkbox"/> Sudo Allow Commands	Delete + Add
<input type="checkbox"/> /opt/third-party-app/bin/report	

☐ Sudo Allow Command Groups [Delete](#) [+ Add](#)

Deny

☐ Sudo Deny Commands [Delete](#) [+ Add](#)

☐ Sudo Deny Command Groups [Delete](#) [+ Add](#)

As Whom

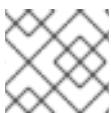
RunAs User category the rule applies to: ☐ Anyone ☒ Specified Users and Groups

<input type="checkbox"/> RunAs Users	External	Delete + Add
<input type="checkbox"/> thirdpartyapp	True	

☐ Groups of RunAs Users [Delete](#) [+ Add](#)

RunAs Group category the rule applies to: ☐ Any Group ☒ Specified Groups

<input type="checkbox"/> RunAs Groups	External	Delete + Add
---------------------------------------	----------	--

**NOTE**

Propagating the changes from the server to the client can take a few minutes.

Verification steps

1. Log in to the **idmclient** host as the **idm_user** account.
2. Test the new sudo rule:
 - a. Display which **sudo** rules the **idm_user** account is allowed to perform.

```
[idm_user@idmclient ~]$ sudo -l
Matching Defaults entries for idm_user@idm.example.com on idmclient:
    !visiblepw, always_set_home, match_group_by_gid, always_query_group_plugin,
```

```
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 KRB5CCNAME",
secure_path="/sbin:/bin:/usr/sbin:/usr/bin
```

User `idm_user@idm.example.com` may run the following commands on `idmclient`:
(thirdpartyapp) /opt/third-party-app/bin/report

- b. Run the **report** command as the **thirdpartyapp** service account.

```
[idm_user@idmclient ~]$ sudo -u thirdpartyapp /opt/third-party-app/bin/report
[sudo] password for idm_user@idm.example.com:
Executing report...
Report successful.
```

18.6. ENABLING GSSAPI AUTHENTICATION FOR SUDO ON AN IDM CLIENT

The following procedure describes enabling GSSAPI authentication on an IdM client for the **sudo** and **sudo -i** commands via the **pam_sss_gss.so** PAM module. This configuration allows IdM users to authenticate to the **sudo** command with their Kerberos ticket.

Prerequisites

- You have created a **sudo** rule for an IdM user that applies to an IdM host. For this example, you have created the **idm_user_reboot sudo** rule to grant the **idm_user** account the permission to run the **/usr/sbin/reboot** command on the **idmclient** host.
- The **idmclient** host is running RHEL 8.4 or later.
- You need **root** privileges to modify the **/etc/sss/sss.conf** file and PAM files in the **/etc/pam.d/** directory.

Procedure

1. Open the **/etc/sss/sss.conf** configuration file.
2. Add the following entry to the **[domain/<domain_name>]** section.

```
[domain/<domain_name>]
pam_gssapi_services = sudo, sudo-i
```

3. Save and close the **/etc/sss/sss.conf** file.
4. Restart the SSSD service to load the configuration changes.

```
[root@idmclient ~]# systemctl restart sssd
```

5. Open the **/etc/pam.d/sudo** PAM configuration file.
6. Add the following entry as the first line of the **auth** section in the **/etc/pam.d/sudo** file.

```
#%PAM-1.0
auth sufficient pam_sss_gss.so
auth    include    system-auth
account include    system-auth
password include    system-auth
session include    system-auth
```

7. Save and close the **/etc/pam.d/sudo** file.
8. Open the **/etc/pam.d/sudo-i** PAM configuration file.
9. Add the following entry as the first line of the **auth** section in the **/etc/pam.d/sudo-i** file.

```
#%PAM-1.0
auth sufficient pam_sss_gss.so
auth    include    sudo
account include    sudo
password include    sudo
session optional    pam_keyinit.so force revoke
session include    sudo
```

10. Save and close the **/etc/pam.d/sudo-i** file.

Verification steps

1. Log into the host as the **idm_user** account.

```
[root@idm-client ~]# ssh -l idm_user@idm.example.com localhost
idm_user@idm.example.com's password:
```

2. Verify that you have a ticket-granting ticket as the **idm_user** account.

```
[idmuser@idmclient ~]$ klist
Ticket cache: KCM:1366201107
Default principal: idm_user@IDM.EXAMPLE.COM

Valid starting    Expires          Service principal
01/08/2021 09:11:48 01/08/2021 19:11:48
krbtgt/IDM.EXAMPLE.COM@IDM.EXAMPLE.COM
renew until 01/15/2021 09:11:44
```

3. (Optional) If you do not have Kerberos credentials for the **idm_user** account, destroy your current Kerberos credentials and request the correct ones.

```
[idm_user@idmclient ~]$ kdestroy -A

[idm_user@idmclient ~]$ kinit idm_user@IDM.EXAMPLE.COM
Password for idm_user@idm.example.com:
```

4. Reboot the machine using **sudo**, without specifying a password.

```
[idm_user@idmclient ~]$ sudo /usr/sbin/reboot
```

Additional resources

- [Granting sudo access to an IdM user on an IdM client using IdM Web UI](#)
- [Granting sudo access to an IdM user on an IdM client using the CLI](#)
- **pam_sss_gss (8)** man page
- **sssd.conf (5)** man page

18.7. ENABLING GSSAPI AUTHENTICATION AND ENFORCING KERBEROS AUTHENTICATION INDICATORS FOR SUDO ON AN IDM CLIENT

The following procedure describes enabling GSSAPI authentication on an IdM client for the **sudo** and **sudo -i** commands via the **pam_sss_gss.so** PAM module. Additionally, only users who have logged in with a smart card will authenticate to those commands with their Kerberos ticket.



NOTE

You can use this procedure as a template to configure GSSAPI authentication with SSSD for other PAM-aware services, and further restrict access to only those users that have a specific authentication indicator attached to their Kerberos ticket.

Prerequisites

- You have created a **sudo** rule for an IdM user that applies to an IdM host. For this example, you have created the **idm_user_reboot sudo** rule to grant the **idm_user** account the permission to run the **/usr/sbin/reboot** command on the **idmclient** host.
- You have configured smart card authentication for the **idmclient** host.
- The **idmclient** host is running RHEL 8.4 or later.
- You need **root** privileges to modify the **/etc/sss/sss.conf** file and PAM files in the **/etc/pam.d/** directory.

Procedure

1. Open the **/etc/sss/sss.conf** configuration file.
2. Add the following entries to the **[domain/<domain_name>]** section.

```
[domain/<domain_name>]
pam_gssapi_services = sudo, sudo-i
pam_gssapi_indicators_map = sudo:pkinit, sudo-i:pkinit
```

3. Save and close the **/etc/sss/sss.conf** file.
4. Restart the SSSD service to load the configuration changes.


```
[root@idmclient ~]# systemctl restart sssd
```

5. Open the **/etc/pam.d/sudo** PAM configuration file.
6. Add the following entry as the first line of the **auth** section in the **/etc/pam.d/sudo** file.

```
#%PAM-1.0
auth sufficient pam_sss_gss.so
auth    include    system-auth
account include    system-auth
password include    system-auth
session include    system-auth
```

7. Save and close the **/etc/pam.d/sudo** file.
8. Open the **/etc/pam.d/sudo-i** PAM configuration file.
9. Add the following entry as the first line of the **auth** section in the **/etc/pam.d/sudo-i** file.

```
#%PAM-1.0
auth sufficient pam_sss_gss.so
auth    include    sudo
account include    sudo
password include    sudo
session optional    pam_keyinit.so force revoke
session include    sudo
```

10. Save and close the **/etc/pam.d/sudo-i** file.

Verification steps

1. Log into the host as the **idm_user** account and authenticate with a smart card.

```
[root@idmclient ~]# ssh -l idm_user@idm.example.com localhost
PIN for smart_card
```

2. Verify that you have a ticket-granting ticket as the smart card user.

```
[idm_user@idmclient ~]$ klist
Ticket cache: KEYRING:persistent:1358900015:krb_cache_TObtNMd
Default principal: idm_user@IDM.EXAMPLE.COM

Valid starting    Expires            Service principal
02/15/2021 16:29:48 02/16/2021 02:29:48
krbtgt/IDM.EXAMPLE.COM@IDM.EXAMPLE.COM
renew until 02/22/2021 16:29:44
```

3. Display which **sudo** rules the **idm_user** account is allowed to perform.

```
[idm_user@idmclient ~]$ sudo -l
Matching Defaults entries for idmuser on idmclient:
    !visiblepw, always_set_home, match_group_by_gid, always_query_group_plugin,
    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
KRB5CCNAME",
secure_path=/sbin\:/bin\:/usr/sbin\:/usr/bin
```

User **idm_user** may run the following commands on **idmclient**:
(root) /usr/sbin/reboot

4. Reboot the machine using **sudo**, without specifying a password.

```
[idm_user@idmclient ~]$ sudo /usr/sbin/reboot
```

Additional resources

- [Configuring Identity Management for smart card authentication](#)
- [Kerberos authentication indicators](#)
- [Granting sudo access to an IdM user on an IdM client using IdM Web UI](#)
- [Granting sudo access to an IdM user on an IdM client using the CLI](#)
- **pam_sss_gss (8)** man page
- **sssd.conf (5)** man page

18.8. SSSD OPTIONS CONTROLLING GSSAPI AUTHENTICATION FOR PAM SERVICES

You can use the following options for the **/etc/sss/sss.conf** configuration file to adjust the GSSAPI configuration within the SSSD service.

pam_gssapi_services

GSSAPI authentication with SSSD is disabled by default. You can use this option to specify a comma-separated list of PAM services that are allowed to try GSSAPI authentication using the **pam_sss_gss.so** PAM module. To explicitly disable GSSAPI authentication, set this option to **-**.

pam_gssapi_indicators_map

This option only applies to Identity Management (IdM) domains. Use this option to list Kerberos authentication indicators that are required to grant PAM access to a service. Pairs must be in the format **<PAM_service>:_<required_authentication_indicator>_**.

Valid authentication indicators are:

- **otp** for two-factor authentication
- **radius** for RADIUS authentication
- **pkinit** for PKINIT, smart card, or certificate authentication
- **hardened** for hardened passwords

pam_gssapi_check_upn

This option is enabled and set to **true** by default. If this option is enabled, the SSSD service requires that the user name matches the Kerberos credentials. If **false**, the **pam_sss_gss.so** PAM module authenticates every user that is able to obtain the required service ticket.

Examples

The following options enable Kerberos authentication for the **sudo** and **sudo-i** services, requires that **sudo** users authenticated with a one-time password, and user names must match the Kerberos principal. Because these settings are in the **[pam]** section, they apply to all domains:

```
[pam]
pam_gssapi_services = sudo, sudo-i
pam_gssapi_indicators_map = sudo:otp
pam_gssapi_check_upn = true
```

You can also set these options in individual **[domain]** sections to overwrite any global values in the **[pam]** section. The following options apply different GSSAPI settings to each domain:

For the **idm.example.com** domain

- Enable GSSAPI authentication for the **sudo** and **sudo -i** services.
- Require certificate or smart card authentication authenticators for the **sudo** command.
- Require one-time password authentication authenticators for the **sudo -i** command.
- Enforce matching user names and Kerberos principals.

For the **ad.example.com** domain

- Enable GSSAPI authentication only for the **sudo** service.
- Do not enforce matching user names and principals.

```
[domain/idm.example.com]
pam_gssapi_services = sudo, sudo-i
pam_gssapi_indicators_map = sudo:pkinit, sudo-i:otp
pam_gssapi_check_upn = true
...
```

```
[domain/ad.example.com]
pam_gssapi_services = sudo
pam_gssapi_check_upn = false
...
```

Additional resources

- [Kerberos authentication indicators](#)

18.9. TROUBLESHOOTING GSSAPI AUTHENTICATION FOR SUDO

If you are unable to authenticate to the **sudo** service with a Kerberos ticket from IdM, use the following scenarios to troubleshoot your configuration.

Scenario 1

Prerequisites

- You have enabled GSSAPI authentication for the **sudo** service. See [Enabling GSSAPI authentication for sudo on an IdM client](#).
- You need **root** privileges to modify the **/etc/sss/sssd.conf** file and PAM files in the **/etc/pam.d/** directory.

Procedure

- If you see the following error, the Kerberos service might not be able to resolve the correct realm for the service ticket based on the host name:

```
Server not found in Kerberos database
```

In this situation, add the hostname directly to **[domain_realm]** section in the **/etc/krb5.conf** Kerberos configuration file:

```
[idm-user@idm-client ~]$ cat /etc/krb5.conf
...

[domain_realm]
.example.com = EXAMPLE.COM
example.com = EXAMPLE.COM
server.example.com = EXAMPLE.COM
```

- If you see the following error, you do not have any Kerberos credentials:

```
No Kerberos credentials available
```

In this situation, retrieve Kerberos credentials with the **kinit** utility or authenticate with SSSD:

```
[idm-user@idm-client ~]$ kinit idm-user@IDM.EXAMPLE.COM
Password for idm-user@idm.example.com:
```

- If you see either of the following errors in the **/var/log/sss/sssd_pam.log** log file, the Kerberos credentials do not match the username of the user currently logged in:

```
User with UPN [<UPN>] was not found.

UPN [<UPN>] does not match target user [<username>].
```

In this situation, verify that you authenticated with SSSD, or consider disabling the **pam_gssapi_check_upn** option in the **/etc/sss/sssd.conf** file:

```
[idm-user@idm-client ~]$ cat /etc/sss/sssd.conf
...

pam_gssapi_check_upn = false
```

- For additional troubleshooting, you can enable debugging output for the **pam_sss_gss.so** PAM module.

- Add the **debug** option at the end of all **pam_sss_gss.so** entries in PAM files, such as **/etc/pam.d/sudo** and **/etc/pam.d/sudo-i**:

```
[root@idm-client ~]# cat /etc/pam.d/sudo
#%PAM-1.0
auth    sufficient pam_sss_gss.so  debug
auth    include     system-auth
account include     system-auth
password include     system-auth
session include     system-auth
```

```
[root@idm-client ~]# cat /etc/pam.d/sudo-i
#%PAM-1.0
auth    sufficient pam_sss_gss.so  debug
auth    include     sudo
account include     sudo
password include     sudo
session optional    pam_keyinit.so force revoke
session include     sudo
```

- Try to authenticate with the **pam_sss_gss.so** module and review the console output. In this example, the user did not have any Kerberos credentials.

```
[idm-user@idm-client ~]$ sudo ls -l /etc/sss/sss.conf
pam_sss_gss: Initializing GSSAPI authentication with SSSD
pam_sss_gss: Switching euid from 0 to 1366201107
pam_sss_gss: Trying to establish security context
pam_sss_gss: SSSD User name: idm-user@idm.example.com
pam_sss_gss: User domain: idm.example.com
pam_sss_gss: User principal:
pam_sss_gss: Target name: host@idm.example.com
pam_sss_gss: Using ccache: KCM:
pam_sss_gss: Acquiring credentials, principal name will be derived
pam_sss_gss: Unable to read credentials from [KCM:] [maj:0xd0000, min:0x96c73ac3]
pam_sss_gss: GSSAPI: Unspecified GSS failure. Minor code may provide more
information
pam_sss_gss: GSSAPI: No credentials cache found
pam_sss_gss: Switching euid from 1366200907 to 0
pam_sss_gss: System error [5]: Input/output error
```

18.10. USING AN ANSIBLE PLAYBOOK TO ENSURE SUDO ACCESS FOR AN IDM USER ON AN IDM CLIENT

In Identity Management (IdM), you can ensure **sudo** access to a specific command is granted to an IdM user account on a specific IdM host.

Complete this procedure to ensure a **sudo** rule named **idm_user_reboot** exists. The rule grants **idm_user** the permission to run the **/usr/sbin/reboot** command on the **idmclient** machine.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller.
- You know the IdM administrator password.

- You have [ensured the presence of a user account for `idm_user` in IdM and unlocked the account by creating a password for the user](#). For details on adding a new IdM user using the command-line interface, see [Adding users using the command line](#).
- No local `idm_user` account exists on `idmclient`. The `idm_user` user is not listed in the `/etc/passwd` file on `idmclient`.

Procedure

1. Create an inventory file, for example **inventory.file**, and define **ipaservers** in it:

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

2. Add one or more **sudo** commands:
 - a. Create an **ensure-reboot-sudocmd-is-present.yml** Ansible playbook that ensures the presence of the `/usr/sbin/reboot` command in the IdM database of **sudo** commands. To simplify this step, you can copy and modify the example in the `/usr/share/doc/ansible-freeipa/playbooks/sudocmd/ensure-sudocmd-is-present.yml` file:

```
---
- name: Playbook to manage sudo command
  hosts: ipaserver
  become: true

  tasks:
    # Ensure sudo command is present
    - ipasudocmd:
        ipadmin_password: MySecret123
        name: /usr/sbin/reboot
        state: present
```

- b. Run the playbook:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
path_to_playbooks_directory/ensure-reboot-sudocmd-is-present.yml
```

3. Create a **sudo** rule that references the commands:
 - a. Create an **ensure-sudorule-for-idmuser-on-idmclient-is-present.yml** Ansible playbook that uses the **sudo** command entry to ensure the presence of a sudo rule. The sudo rule allows `idm_user` to reboot the `idmclient` machine. To simplify this step, you can copy and modify the example in the `/usr/share/doc/ansible-freeipa/playbooks/sudorule/ensure-sudorule-is-present.yml` file:

```
---
- name: Tests
  hosts: ipaserver
  become: true

  tasks:
    # Ensure a sudorule is present granting idm_user the permission to run /usr/sbin/reboot
    on idmclient
    - ipasudorule:
```

```

ipaadmin_password: MySecret123
name: idm_user_reboot
description: A test sudo rule.
allow_sudocmd: /usr/sbin/reboot
host: idmclient.idm.example.com
user: idm_user
state: present

```

- b. Run the playbook:

```

$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
path_to_playbooks_directory/ensure-sudorule-for-idmuser-on-idmclient-is-
present.yml

```

Verification steps

Test that the **sudo** rule whose presence you have ensured on the IdM server works on **idmclient** by verifying that **idm_user** can reboot **idmclient** using **sudo**. Note that it can take a few minutes for the changes made on the server to take effect on the client.

1. Log in to **idmclient** as **idm_user**.
2. Reboot the machine using **sudo**. Enter the password for **idm_user** when prompted:

```

$ sudo /usr/sbin/reboot
[sudo] password for idm_user:

```

If **sudo** is configured correctly, the machine reboots.

Additional materials

- For more details on how to apply **sudo** commands, command groups, and rules in IdM using an Ansible playbook including the descriptions of playbook variables, see the README-sudocmd.md, README-sudocmdgroup.md, and README-sudorule.md Markdown files available in the **/usr/share/doc/ansible-freeipa/** directory.

CHAPTER 19. ENSURING THE PRESENCE OF HOST-BASED ACCESS CONTROL RULES IN IDM USING ANSIBLE PLAYBOOKS

This chapter describes Identity Management (IdM) host-based access policies and how to define them using [Ansible](#).

Ansible is an automation tool used to configure systems, deploy software, and perform rolling updates. It includes support for Identity Management (IdM).

19.1. HOST-BASED ACCESS CONTROL RULES IN IDM

Host-based access control (HBAC) rules define which users or user groups can access which hosts or host groups by using which services or services in a service group. As a system administrator, you can use HBAC rules to achieve the following goals:

- Limit access to a specified system in your domain to members of a specific user group.
- Allow only a specific service to be used to access systems in your domain.

By default, IdM is configured with a default HBAC rule named **allow_all**, which means universal access to every host for every user via every relevant service in the entire IdM domain.

You can fine-tune access to different hosts by replacing the default **allow_all** rule with your own set of HBAC rules. For centralized and simplified access control management, you can apply HBAC rules to user groups, host groups, or service groups instead of individual users, hosts, or services.

19.2. ENSURING THE PRESENCE OF AN HBAC RULE IN IDM USING AN ANSIBLE PLAYBOOK

This section describes how to ensure the presence of a host-based access control (HBAC) rule in Identity Management (IdM) using an Ansible playbook.

Prerequisites

- The [ansible-freeipa](#) package is installed on the Ansible controller.
- You know the IdM administrator password.
- The users and user groups you want to use for your HBAC rule exist in IdM. See [Managing user accounts using Ansible playbooks](#) and [Ensuring the presence of IdM groups and group members using Ansible playbooks](#) for details.
- The hosts and host groups to which you want to apply your HBAC rule exist in IdM. See [Managing hosts using Ansible playbooks](#) and [Managing host groups using Ansible playbooks](#) for details.

Procedure

1. Create an inventory file, for example **inventory.file**, and define **ipaserver** in it:

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


2. Create your Ansible playbook file that defines the HBAC policy whose presence you want to ensure. To simplify this step, you can copy and modify the example in the **/usr/share/doc/ansible-freeipa/playbooks/hbacrule/ensure-hbacrule-allhosts-present.yml** file:

```
---
- name: Playbook to handle hbacrules
  hosts: ipaserver
  become: true

  tasks:
    # Ensure idm_user can access client.idm.example.com via the sshd service
    - ipahbacrule:
        ipadmin_password: MySecret123
        name: login
        user: idm_user
        host: client.idm.example.com
        hbacsvc:
          - sshd
        state: present
```

3. Run the playbook:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
path_to_playbooks_directory/ensure-new-hbacrule-present.yml
```

Verification steps

1. Log in to the IdM Web UI as administrator.
2. Navigate to **Policy → Host-Based-Access-Control → HBAC Test**
3. In the **Who** tab, select **idm_user**.
4. In the **Accessing** tab, select **client.idm.example.com**.
5. In the **Via service** tab, select **sshd**.
6. In the **Rules** tab, select **login**.
7. In the **Run test** tab, click the **Run test** button. If you see **ACCESS GRANTED**, the HBAC rule is implemented successfully.

Additional resources

- For more details about and examples of, configuring HBAC services, service groups, and rules using Ansible, see the **README-hbacsvc.md**, **README-hbacsvcgroup.md**, and **README-hbacrule.md** Markdown files. These files are available in the **/usr/share/doc/ansible-freeipa** directory. Also see the playbooks available in the relevant subdirectories of the **/usr/share/doc/ansible-freeipa/playbooks** directory.

CHAPTER 20. VAULTS IN IDM

This chapter describes vaults in Identity Management (IdM). It introduces the following topics:

- [The concept of the vault.](#)
- [The different roles associated with a vault .](#)
- [The different types of vaults available in IdM based on the level of security and access control .](#)
- [The different types of vaults available in IdM based on ownership .](#)
- [The concept of vault containers.](#)
- [The basic commands for managing vaults in IdM .](#)
- [Installing the key recovery authority \(KRA\), which is a prerequisite for using vaults in IdM .](#)

20.1. VAULTS AND THEIR BENEFITS

A vault is a useful feature for those Identity Management (IdM) users who want to keep all their sensitive data stored securely but conveniently in one place. This section explains the various types of vaults and their uses, and which vault you should choose based on your requirements.

A vault is a secure location in (IdM) for storing, retrieving, sharing, and recovering a secret. A secret is security-sensitive data, usually authentication credentials, that only a limited group of people or entities can access. For example, secrets include:

- passwords
- PINs
- private SSH keys

A vault is comparable to a password manager. Just like a password manager, a vault typically requires a user to generate and remember one primary password to unlock and access any information stored in the vault. However, a user can also decide to have a standard vault. A standard vault does not require the user to enter any password to access the secrets stored in the vault.



NOTE

The purpose of vaults in IdM is to store authentication credentials that allow you to authenticate to external, non-IdM-related services.

Other important characteristics of the IdM vaults are:

- Vaults are only accessible to the vault owner and those IdM users that the vault owner selects to be the vault members. In addition, the IdM administrator has access to the vault.
- If a user does not have sufficient privileges to create a vault, an IdM administrator can create the vault and set the user as its owner.
- Users and services can access the secrets stored in a vault from any machine enrolled in the IdM domain.

- One vault can only contain one secret, for example, one file. However, the file itself can contain multiple secrets such as passwords, keytabs or certificates.



NOTE

Vault is only available from the IdM command line (CLI), not from the IdM Web UI.

20.2. VAULT OWNERS, MEMBERS, AND ADMINISTRATORS

Identity Management (IdM) distinguishes the following vault user types:

Vault owner

A vault owner is a user or service with basic management privileges on the vault. For example, a vault owner can modify the properties of the vault or add new vault members.

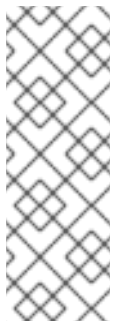
Each vault must have at least one owner. A vault can also have multiple owners.

Vault member

A vault member is a user or service that can access a vault created by another user or service.

Vault administrator

Vault administrators have unrestricted access to all vaults and are allowed to perform all vault operations.



NOTE

Symmetric and asymmetric vaults are protected with a password or key and apply special access control rules (see [Vault types](#)). The administrator must meet these rules to:

- Access secrets in symmetric and asymmetric vaults.
- Change or reset the vault password or key.

A vault administrator is any user with the **Vault Administrators** privilege. In the context of the role-based access control (RBAC) in IdM, a privilege is a group of permissions that you can apply to a role.

Vault User

The vault user represents the user in whose container the vault is located. The **Vault user** information is displayed in the output of specific commands, such as **ipa vault-show**:

```
$ ipa vault-show my_vault
Vault name: my_vault
Type: standard
Owner users: user
Vault user: user
```

For details on vault containers and user vaults, see [Vault containers](#).

Additional resources

- Certain owner and member privileges depend on the type of the vault. See [Standard, symmetric and asymmetric vaults](#) for details.

20.3. STANDARD, SYMMETRIC, AND ASYMMETRIC VAULTS

Based on the level of security and access control, IdM classifies vaults into the following types:

Standard vaults

Vault owners and vault members can archive and retrieve the secrets without having to use a password or key.

Symmetric vaults

Secrets in the vault are protected with a symmetric key. Vault owners and members can archive and retrieve the secrets, but they must provide the vault password.

Asymmetric vaults

Secrets in the vault are protected with an asymmetric key. Users archive the secret using a public key and retrieve it using a private key. Vault members can only archive secrets, while vault owners can do both, archive and retrieve secrets.

20.4. USER, SERVICE, AND SHARED VAULTS

Based on ownership, IdM classifies vaults into several types. The [table below](#) contains information about each type, its owner and use.

Table 20.1. IdM vaults based on ownership

Type	Description	Owner	Note
User vault	A private vault for a user	A single user	Any user can own one or more user vaults if allowed by IdM administrator
Service vault	A private vault for a service	A single service	Any service can own one or more user vaults if allowed by IdM administrator
Shared vault	A vault shared by multiple users and services	The vault administrator who created the vault	Users and services can own one or more user vaults if allowed by IdM administrator. The vault administrators other than the one that created the vault also have full access to the vault.

20.5. VAULT CONTAINERS

A vault container is a collection of vaults. The [table below](#) lists the default vault containers that Identity Management (IdM) provides.

Table 20.2. Default vault containers in IdM

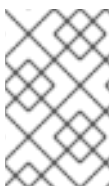
Type	Description	Purpose
User container	A private container for a user	Stores user vaults for a particular user
Service container	A private container for a service	Stores service vaults for a particular service

Type	Description	Purpose
Shared container	A container for multiple users and services	Stores vaults that can be shared by multiple users or services

IdM creates user and service containers for each user or service automatically when the first private vault for the user or service is created. After the user or service is deleted, IdM removes the container and its contents.

20.6. BASIC IDM VAULT COMMANDS

This section describes basic commands you can use to manage Identity Management (IdM) vaults. The [table below](#) contains a list of **ipa vault-*** commands with the explanation of their purpose.



NOTE

Before running any **ipa vault-*** command, install the Key Recovery Authority (KRA) certificate system component on one or more of the servers in your IdM domain. For details, see [Installing the Key Recovery Authority in IdM](#).

Table 20.3. Basic IdM vault commands with explanations

Command	Purpose
ipa help vault	Displays conceptual information about IdM vaults and sample vault commands.
ipa vault-add --help , ipa vault-find --help	Adding the --help option to a specific ipa vault-* command displays the options and detailed help available for that command.
ipa vault-show user_vault --user idm_user	When accessing a vault as a vault member, you must specify the vault owner. If you do not specify the vault owner, IdM informs you that it did not find the vault: <pre>[admin@server ~]\$ ipa vault-show user_vault ipa: ERROR: user_vault: vault not found</pre>
ipa vault-show shared_vault --shared	When accessing a shared vault, you must specify that the vault you want to access is a shared vault. Otherwise, IdM informs you it did not find the vault: <pre>[admin@server ~]\$ ipa vault-show shared_vault ipa: ERROR: shared_vault: vault not found</pre>

20.7. INSTALLING THE KEY RECOVERY AUTHORITY IN IDM

This section describes how you can enable vaults in Identity Management (IdM) by installing the Key Recovery Authority (KRA) Certificate System (CS) component.

Prerequisites

- You are logged in as IdM administrator.

- You are logged in as root on an IdM client.

Procedure

- Install the KRA:

```
# ipa-kra-install
```



IMPORTANT

You can install the first KRA of an IdM cluster on a hidden replica. However, installing additional KRAs requires temporarily activating the hidden replica before you install the KRA clone on a non-hidden replica. Then you can hide the originally hidden replica again.



NOTE

To make the vault service highly available, install the KRA on two IdM servers or more.

Additional resources

- For more information on how to activate an IdM replica and how to hide it, see [Demoting or promoting hidden replicas](#).
- For more information on hidden replicas in IdM, see [The hidden replica mode](#).

CHAPTER 21. USING ANSIBLE TO MANAGE IDM USER VAULTS: STORING AND RETRIEVING SECRETS

This chapter describes how to manage user vaults in Identity Management using the Ansible **vault** module. Specifically, it describes how a user can use Ansible playbooks to perform the following three consecutive actions:

- [Create an user vault in IdM](#) .
- [Store a secret in the vault](#) .
- [Retrieve a secret from the vault](#) .

The user can do the storing and the retrieving from two different IdM clients.

Prerequisites

- The Key Recovery Authority (KRA) Certificate System component has been installed on one or more of the servers in your IdM domain. For details, see [Installing the Key Recovery Authority in IdM](#).

21.1. ENSURING THE PRESENCE OF A STANDARD USER VAULT IN IDM USING ANSIBLE

This section shows how an Identity Management (IdM) user can use an Ansible playbook to create a vault container with one or more private vaults to securely store sensitive information. In the example used in the procedure below, the **idm_user** user creates a vault of the standard type named **my_vault**. The standard vault type ensures that **idm_user** will not be required to authenticate when accessing the file. **idm_user** will be able to retrieve the file from any IdM client to which the user is logged in.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller, that is the host on which you execute the steps in the procedure.
- You know the password of **idm_user**.

Procedure

1. Navigate to the **/usr/share/doc/ansible-freeipa/playbooks/vault** directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/vault
```

2. Create an inventory file, for example **inventory.file**:

```
$ touch inventory.file
```

3. Open **inventory.file** and define the IdM server that you want to configure in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

4. Make a copy of the **ensure-standard-vault-is-present.yml** Ansible playbook file. For example:

```
$ cp ensure-standard-vault-is-present.yml ensure-standard-vault-is-present-copy.yml
```

5. Open the **ensure-standard-vault-is-present-copy.yml** file for editing.
6. Adapt the file by setting the following variables in the **ipavault** task section:

- Set the **ipaadmin_principal** variable to **idm_user**.
- Set the **ipaadmin_password** variable to the password of **idm_user**.
- Set the **user** variable to **idm_user**.
- Set the **name** variable to **my_vault**.
- Set the **vault_type** variable to **standard**.

This is the modified Ansible playbook file for the current example:

```
---
- name: Tests
  hosts: ipaserver
  become: true
  gather_facts: false

  tasks:
  - ipavault:
    ipaadmin_principal: idm_user
    ipaadmin_password: idm_user_password
    user: idm_user
    name: my_vault
    vault_type: standard
```

7. Save the file.
8. Run the playbook:

```
$ ansible-playbook -v -i inventory.file ensure-standard-vault-is-present-copy.yml
```

21.2. ARCHIVING A SECRET IN A STANDARD USER VAULT IN IDM USING ANSIBLE

This section shows how an Identity Management (IdM) user can use an Ansible playbook to store sensitive information in a personal vault. In the example used, the **idm_user** user archives a file with sensitive information named **password.txt** in a vault named **my_vault**.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller, that is the host on which you execute the steps in the procedure.
- You know the password of **idm_user**.
- **idm_user** is the owner, or at least a member user of **my_vault**.

- You have access to **password.txt**, the secret that you want to archive in **my_vault**.

Procedure

1. Navigate to the **/usr/share/doc/ansible-freeipa/playbooks/vault** directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/vault
```

2. Open your inventory file and make sure that the IdM server that you want to configure is listed in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

3. Make a copy of the **data-archive-in-symmetric-vault.yml** Ansible playbook file but replace "symmetric" by "standard". For example:

```
$ cp data-archive-in-symmetric-vault.yml data-archive-in-standard-vault-copy.yml
```

4. Open the **data-archive-in-standard-vault-copy.yml** file for editing.
5. Adapt the file by setting the following variables in the **ipavault** task section:

- Set the **ipaadmin_principal** variable to **idm_user**.
 - Set the **ipaadmin_password** variable to the password of **idm_user**.
 - Set the **user** variable to **idm_user**.
 - Set the **name** variable to **my_vault**.
 - Set the **in** variable to the full path to the file with sensitive information.
 - Set the **action** variable to **member**.
- This is the modified Ansible playbook file for the current example:

```
---
- name: Tests
  hosts: ipaserver
  become: true
  gather_facts: false

  tasks:
  - ipavault:
    ipaadmin_principal: idm_user
    ipaadmin_password: idm_user_password
    user: idm_user
    name: my_vault
    in: /usr/share/doc/ansible-freeipa/playbooks/vault/password.txt
    action: member
```

6. Save the file.
7. Run the playbook:

```
$ ansible-playbook -v -i inventory.file data-archive-in-standard-vault-copy.yml
```

21.3. RETRIEVING A SECRET FROM A STANDARD USER VAULT IN IDM USING ANSIBLE

This section shows how an Identity Management (IdM) user can use an Ansible playbook to retrieve a secret from the user personal vault. In the example used in the procedure below, the **idm_user** user retrieves a file with sensitive data from a vault of the standard type named **my_vault** onto an IdM client named **host01**. **idm_user** does not have to authenticate when accessing the file. **idm_user** can use Ansible to retrieve the file from any IdM client on which Ansible is installed.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller. This is the host on which you execute the steps in the procedure.
- You know the password of **idm_user**.
- **idm_user** is the owner of **my_vault**.
- **idm_user** has stored a secret in **my_vault**.
- Ansible can write into the directory on the IdM host into which you want to retrieve the secret.
- **idm_user** can read from the directory on the IdM host into which you want to retrieve the secret.

Procedure

1. Navigate to the **/usr/share/doc/ansible-freeipa/playbooks/vault** directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/vault
```

2. Open your inventory file and mention, in a clearly defined section, the IdM client onto which you want to retrieve the secret. For example, to instruct Ansible to retrieve the secret onto **host01.idm.example.com**, enter:

```
[ipahost]
host01.idm.example.com
```

3. Make a copy of the **retrive-data-symmetric-vault.yml** Ansible playbook file. Replace "symmetric" with "standard". For example:

```
$ cp retrive-data-symmetric-vault.yml retrieve-data-standard-vault.yml-copy.yml
```

4. Open the **retrieve-data-standard-vault.yml-copy.yml** file for editing.
5. Adapt the file by setting the **hosts** variable to **ipahost**.
6. Adapt the file by setting the following variables in the **ipavault** task section:
 - Set the **ipaadmin_principal** variable to **idm_user**.
 - Set the **ipaadmin_password** variable to the password of **idm_user**.

- Set the **user** variable to **idm_user**.
- Set the **name** variable to **my_vault**.
- Set the **out** variable to the full path of the file into which you want to export the secret.
- Set the **state** variable to **retrieved**.

This the modified Ansible playbook file for the current example:

```
---
- name: Tests
  hosts: ipahost
  become: true
  gather_facts: false

  tasks:
  - ipavault:
      ipaadmin_principal: idm_user
      ipaadmin_password: idm_user_password
      user: idm_user
      name: my_vault
      out: /tmp/password_exported.txt
      state: retrieved
```

7. Save the file.

8. Run the playbook:

```
$ ansible-playbook -v -i inventory.file retrieve-data-standard-vault.yml-copy.yml
```

Verification steps

1. **SSH** to **host01** as **user01**:

```
$ ssh user01@host01.idm.example.com
```

2. View the file specified by the **out** variable in the Ansible playbook file:

```
$ vim /tmp/password_exported.txt
```

You can now see the exported secret.

- For more information about using Ansible to manage IdM vaults and user secrets and about playbook variables, see the README-vault.md Markdown file available in the **/usr/share/doc/ansible-freeipa/** directory and the sample playbooks available in the **/usr/share/doc/ansible-freeipa/playbooks/vault/** directory.

CHAPTER 22. USING ANSIBLE TO MANAGE IDM SERVICE VAULTS: STORING AND RETRIEVING SECRETS

This section shows how an administrator can use the **ansible-freeipa vault** module to securely store a service secret in a centralized location.

The **vault** used in the example is asymmetric, which means that in order to use it, the administrator needs to perform the following steps:

1. Generate a private key using, for example, the **openssl** utility.
2. Generate a public key based on the private key.

The service secret is encrypted with the public key when an administrator archives it into the vault. Afterwards, a service instance hosted on a specific machine in the domain retrieves the secret using the private key. Only the service and the administrator are allowed to access the secret.

If the secret is compromised, the administrator can replace it in the service vault and then redistribute it to those individual service instances that have not been compromised.

Prerequisites

- The Key Recovery Authority (KRA) Certificate System component has been installed on one or more of the servers in your IdM domain. For details, see [Installing the Key Recovery Authority in IdM](#).

This section includes these procedures:

- [Ensuring the presence of an asymmetric service vault in IdM using Ansible](#)
- [Storing an IdM service secret in an asymmetric vault using Ansible](#)
- [Retrieving a service secret for an IdM service using Ansible](#)
- [Changing an IdM service vault secret when compromised using Ansible](#)

In the procedures:

- **admin** is the administrator who manages the service password.
- **private-key-to-an-externally-signed-certificate.pem** is the file containing the service secret, in this case a private key to an externally signed certificate. Do not confuse this private key with the private key used to retrieve the secret from the vault.
- **secret_vault** is the vault created to store the service secret.
- **HTTP/webserver1.idm.example.com** is the service that is the owner of the vault.
- **HTTP/webserver2.idm.example.com** and **HTTP/webserver3.idm.example.com** are the vault member services.
- **service-public.pem** is the service public key used to encrypt the password stored in **password_vault**.
- **service-private.pem** is the service private key used to decrypt the password stored in **secret_vault**.

22.1. ENSURING THE PRESENCE OF AN ASYMMETRIC SERVICE VAULT IN IDM USING ANSIBLE

This section shows how an Identity Management (IdM) administrator can use an Ansible playbook to create a service vault container with one or more private vaults to securely store sensitive information. In the example used in the procedure below, the administrator creates an asymmetric vault named **secret_vault**. This ensures that the vault members have to authenticate using a private key in order to retrieve the secret in the vault. The vault members will be able to retrieve the file from any IdM client.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller. This is the host on which you execute the steps in the procedure.
- You know the **IdM administrator** password.

Procedure

1. Navigate to the **/usr/share/doc/ansible-freeipa/playbooks/vault** directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/vault
```

2. Obtain the public key of the service instance. For example, using the **openssl** utility:
 - a. Generate the **service-private.pem** private key.

```
$ openssl genrsa -out service-private.pem 2048
Generating RSA private key, 2048 bit long modulus
.+++
.....+++
e is 65537 (0x10001)
```

- b. Generate the **service-public.pem** public key based on the private key.

```
$ openssl rsa -in service-private.pem -out service-public.pem -pubout
writing RSA key
```

3. Optional: Create an inventory file if it does not exist, for example **inventory.file**:

```
$ touch inventory.file
```

4. Open your inventory file and define the IdM server that you want to configure in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

5. Make a copy of the **ensure-asymmetric-vault-is-present.yml** Ansible playbook file. For example:

```
$ cp ensure-asymmetric-vault-is-present.yml ensure-asymmetric-service-vault-is-present-copy.yml
```

6. Open the **ensure-asymmetric-vault-is-present-copy.yml** file for editing.
7. Add a task that copies the **service-public.pem** public key from the Ansible controller to the **server.idm.example.com** server.
8. Modify the rest of the file by setting the following variables in the **ipavault** task section:
 - Set the **ipaadmin_password** variable to the IdM administrator password.
 - Define the name of the vault using the **name** variable, for example **secret_vault**.
 - Set the **vault_type** variable to **asymmetric**.
 - Set the **service** variable to the principal of the service that owns the vault, for example **HTTP/webserver1.idm.example.com**.
 - Set the **public_key_file** to the location of your public key.

This is the modified Ansible playbook file for the current example:

```

---
- name: Tests
  hosts: ipaserver
  become: true
  gather_facts: false
  tasks:
    - name: Copy public key to ipaserver.
      copy:
        src: /path/to/service-public.pem
        dest: /usr/share/doc/ansible-freeipa/playbooks/vault/service-public.pem
        mode: 0600
    - name: Add data to vault, from a LOCAL file.
      ipavault:
        ipaadmin_password: Secret123
        name: secret_vault
        vault_type: asymmetric
        service: HTTP/webserver1.idm.example.com
        public_key_file: /usr/share/doc/ansible-freeipa/playbooks/vault/service-public.pem

```

9. Save the file.
10. Run the playbook:

```
$ ansible-playbook -v -i inventory.file ensure-asymmetric-service-vault-is-present-copy.yml
```

22.2. ADDING MEMBER SERVICES TO AN ASYMMETRIC VAULT USING ANSIBLE

This section shows how an Identity Management (IdM) administrator can use an Ansible playbook to add member services to a service vault so that they can all retrieve the secret stored in the vault. In the example used in the procedure below, the IdM administrator adds the **HTTP/webserver2.idm.example.com** and **HTTP/webserver3.idm.example.com** service principals to the **secret_vault** vault that is owned by **HTTP/webserver1.idm.example.com**.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller. This is the host on which you execute the steps in the procedure.
- You know the **IdM administrator** password.
- You have [created an asymmetric vault](#) to store the service secret.

Procedure

1. Navigate to the **/usr/share/doc/ansible-freeipa/playbooks/vault** directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/vault
```

2. Optional: Create an inventory file if it does not exist, for example **inventory.file**:

```
$ touch inventory.file
```

3. Open your inventory file and define the IdM server that you want to configure in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

4. Make a copy of the **data-archive-in-asymmetric-vault.yml** Ansible playbook file. For example:

```
$ cp data-archive-in-asymmetric-vault.yml add-services-to-an-asymmetric-vault.yml
```

5. Open the **data-archive-in-asymmetric-vault-copy.yml** file for editing.

6. Modify the file by setting the following variables in the **ipavault** task section:

- Set the **ipaadmin_password** variable to the IdM administrator password.
- Set the **name** variable to the name of the vault, for example **secret_vault**.
- Set the **service** variable to the service owner of the vault, for example **HTTP/webserver1.idm.example.com**.
- Define the services that you want to have access to the vault secret using the **services** variable.
- Set the **action** variable to **member**.
This is the modified Ansible playbook file for the current example:

```
---
- name: Tests
  hosts: ipaserver
  become: true
  gather_facts: false

  tasks:
  - ipavault:
    ipaadmin_password: Secret123
```

```

name: secret_vault
service: HTTP/webserver1.idm.example.com
services:
- HTTP/webserver2.idm.example.com
- HTTP/webserver3.idm.example.com
action: member

```

7. Save the file.
8. Run the playbook:

```
$ ansible-playbook -v -i inventory.file add-services-to-an-asymmetric-vault.yml
```

22.3. STORING AN IDM SERVICE SECRET IN AN ASYMMETRIC VAULT USING ANSIBLE

This section shows how an Identity Management (IdM) administrator can use an Ansible playbook to store a secret in a service vault so that it can be later retrieved by the service. In the example used in the procedure below, the administrator stores a **PEM** file with the secret in an asymmetric vault named **secret_vault**. This ensures that the service will have to authenticate using a private key in order to retrieve the secret from the vault. The vault members will be able to retrieve the file from any IdM client.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller. This is the host on which you execute the steps in the procedure.
- You know the **IdM administrator** password.
- You have [created an asymmetric vault](#) to store the service secret.
- The secret is stored locally on the Ansible controller, for example in the `/usr/share/doc/ansible-freeipa/playbooks/vault/private-key-to-an-externally-signed-certificate.pem` file.

Procedure

1. Navigate to the `/usr/share/doc/ansible-freeipa/playbooks/vault` directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/vault
```

2. Optional: Create an inventory file if it does not exist, for example **inventory.file**:

```
$ touch inventory.file
```

3. Open your inventory file and define the IdM server that you want to configure in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

4. Make a copy of the **data-archive-in-asymmetric-vault.yml** Ansible playbook file. For example:


```
$ cp data-archive-in-asymmetric-vault.yml data-archive-in-asymmetric-vault-copy.yml
```

5. Open the **data-archive-in-asymmetric-vault-copy.yml** file for editing.
6. Modify the file by setting the following variables in the **ipavault** task section:
 - Set the **ipaadmin_password** variable to the IdM administrator password.
 - Set the **name** variable to the name of the vault, for example **secret_vault**.
 - Set the **service** variable to the service owner of the vault, for example **HTTP/webserver1.idm.example.com**.
 - Set the **in** variable to **"{{ lookup('file', 'private-key-to-an-externally-signed-certificate.pem') | b64encode }}"**. This ensures that Ansible retrieves the file with the private key from the working directory on the Ansible controller rather than from the IdM server.
 - Set the **action** variable to **member**.

This is the modified Ansible playbook file for the current example:

```
---
- name: Tests
  hosts: ipaserver
  become: true
  gather_facts: false

  tasks:
  - ipavault:
    ipaadmin_password: Secret123
    name: secret_vault
    service: HTTP/webserver1.idm.example.com
    in: "{{ lookup('file', 'private-key-to-an-externally-signed-certificate.pem') | b64encode }}"
    action: member
```

7. Save the file.
8. Run the playbook:

```
$ ansible-playbook -v -i inventory.file data-archive-in-asymmetric-vault-copy.yml
```

22.4. RETRIEVING A SERVICE SECRET FOR AN IDM SERVICE USING ANSIBLE

This section shows how an Identity Management (IdM) user can use an Ansible playbook to retrieve a secret from a service vault on behalf of the service. In the example used in the procedure below, running the playbook retrieves a **PEM** file with the secret from an asymmetric vault named **secret_vault**, and stores it in the specified location on all the hosts listed in the Ansible inventory file as **ipaservers**.

The services authenticate to IdM using keytabs, and they authenticate to the vault using a private key. You can retrieve the file on behalf of the service from any IdM client on which **ansible-freeipa** is installed.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller. This is the host on which you execute the steps in the procedure.
- You know the IdM administrator password.
- You have [created an asymmetric vault](#) to store the service secret.
- You have [archived the secret in the vault](#).
- You have stored the private key used to retrieve the service vault secret in the location specified by the **private_key_file** variable on the Ansible controller.

Procedure

1. Navigate to the **/usr/share/doc/ansible-freeipa/playbooks/vault** directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/vault
```

2. Optional: Create an inventory file if it does not exist, for example **inventory.file**:

```
$ touch inventory.file
```

3. Open your inventory file and define the following hosts:
 - Define your IdM server in the **[ipaserver]** section.
 - Define the hosts onto which you want to retrieve the secret in the **[webservers]** section. For example, to instruct Ansible to retrieve the secret to **webserver1.idm.example.com**, **webserver2.idm.example.com**, and **webserver3.idm.example.com**, enter:

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

[webservers]
webserver1.idm.example.com
webserver2.idm.example.com
webserver3.idm.example.com
```

4. Make a copy of the **retrieve-data-asymmetric-vault.yml** Ansible playbook file. For example:

```
$ cp retrieve-data-asymmetric-vault.yml retrieve-data-asymmetric-vault-copy.yml
```

5. Open the **retrieve-data-asymmetric-vault-copy.yml** file for editing.
6. Modify the file by setting the following variables in the **ipavault** task section:
 - Set the **ipaadmin_password** variable to your IdM administrator password.
 - Set the **name** variable to the name of the vault, for example **secret_vault**.
 - Set the **service** variable to the service owner of the vault, for example **HTTP/webserver1.idm.example.com**.
 - Set the **private_key_file** variable to the location of the private key used to retrieve the service vault secret.

- Set the **out** variable to the location on the IdM server where you want to retrieve the **private-key-to-an-externally-signed-certificate.pem** secret, for example the current working directory.
- Set the **action** variable to **member**.
This the modified Ansible playbook file for the current example:

```
---
- name: Retrieve data from vault
  hosts: ipaserver
  become: no
  gather_facts: false

  tasks:
  - name: Retrieve data from the service vault
    ipavault:
      ipaadmin_password: Secret123
      name: secret_vault
      service: HTTP/webserver1.idm.example.com
      vault_type: asymmetric
      private_key: "{{ lookup('file', 'service-private.pem') | b64encode }}"
      out: private-key-to-an-externally-signed-certificate.pem
      state: retrieved
```

7. Add a section to the playbook that retrieves the data file from the IdM server to the Ansible controller:

```
---
- name: Retrieve data from vault
  hosts: ipaserver
  become: no
  gather_facts: false
  tasks:
  [...]
  - name: Retrieve data file
    fetch:
      src: private-key-to-an-externally-signed-certificate.pem
      dest: ./
      flat: yes
      mode: 0600
```

8. Add a section to the playbook that transfers the retrieved **private-key-to-an-externally-signed-certificate.pem** file from the Ansible controller on to the webserver listed in the **webserver** section of the inventory file:

```
---
- name: Send data file to webserver
  become: no
  gather_facts: no
  hosts: webserver
  tasks:
  - name: Send data to webserver
    copy:
```

```
src: private-key-to-an-externally-signed-certificate.pem
dest: /etc/pki/tls/private/httpd.key
mode: 0444
```

9. Save the file.

10. Run the playbook:

```
$ ansible-playbook -v -i inventory.file retrieve-data-asymmetric-vault-copy.yml
```

22.5. CHANGING AN IDM SERVICE VAULT SECRET WHEN COMPROMISED USING ANSIBLE

This section shows how an Identity Management (IdM) administrator can reuse an Ansible playbook to change the secret stored in a service vault when a service instance has been compromised. The scenario in the following example assumes that on **webserver3.idm.example.com**, the retrieved secret has been compromised, but not the key to the asymmetric vault storing the secret. In the example, the administrator reuses the Ansible playbooks used when [storing a secret in an asymmetric vault](#) and [retrieving a secret from the asymmetric vault onto IdM hosts](#). At the start of the procedure, the IdM administrator stores a new **PEM** file with a new secret in the asymmetric vault, adapts the inventory file so as not to retrieve the new secret on to the compromised web server, **webserver3.idm.example.com**, and then re-runs the two procedures.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller. This is the host on which you execute the steps in the procedure.
- You know the **IdM administrator** password.
- You have [created an asymmetric vault](#) to store the service secret.
- You have generated a new **httpd** key for the web services running on IdM hosts to replace the compromised old key.
- The new **httpd** key is stored locally on the Ansible controller, for example in the `/usr/share/doc/ansible-freeipa/playbooks/vault/private-key-to-an-externally-signed-certificate.pem` file.

Procedure

1. Navigate to the `/usr/share/doc/ansible-freeipa/playbooks/vault` directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/vault
```

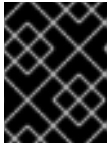
2. Open your inventory file and make sure that the following hosts are defined correctly:

- The IdM server in the **[ipaserver]** section.
- The hosts onto which you want to retrieve the secret in the **[webservers]** section. For example, to instruct Ansible to retrieve the secret to **webserver1.idm.example.com** and **webserver2.idm.example.com**, enter:

```
[ipaserver]
```

```
server.idm.example.com

[webservers]
webserver1.idm.example.com
webserver2.idm.example.com
```



IMPORTANT

Make sure that the list does not contain the compromised webserver, in the current example **webserver3.idm.example.com**.

3. Open the **data-archive-in-asymmetric-vault-copy.yml** file for editing.
 4. Modify the file by setting the following variables in the **ipavault** task section:
 - Set the **ipaadmin_password** variable to the IdM administrator password.
 - Set the **name** variable to the name of the vault, for example **secret_vault**.
 - Set the **service** variable to the service owner of the vault, for example **HTTP/webserver.idm.example.com**.
 - Set the **in** variable to **"{{ lookup('file', 'new-private-key-to-an-externally-signed-certificate.pem') | b64encode }}"**. This ensures that Ansible retrieves the file with the private key from the working directory on the Ansible controller rather than from the IdM server.
 - Set the **action** variable to **member**.
- This is the modified Ansible playbook file for the current example:

```
---
- name: Tests
  hosts: ipaserver
  become: true
  gather_facts: false

  tasks:
  - ipavault:
      ipaadmin_password: Secret123
      name: secret_vault
      service: HTTP/webserver.idm.example.com
      in: "{{ lookup('file', 'new-private-key-to-an-externally-signed-certificate.pem') | b64encode
      }}"
      action: member
```

5. Save the file.
6. Run the playbook:

```
$ ansible-playbook -v -i inventory.file data-archive-in-asymmetric-vault-copy.yml
```

7. Open the **retrieve-data-asymmetric-vault-copy.yml** file for editing.
8. Modify the file by setting the following variables in the **ipavault** task section:

- Set the **ipaadmin_password** variable to your IdM administrator password.
 - Set the **name** variable to the name of the vault, for example **secret_vault**.
 - Set the **service** variable to the service owner of the vault, for example **HTTP/webserver1.idm.example.com**.
 - Set the **private_key_file** variable to the location of the private key used to retrieve the service vault secret.
 - Set the **out** variable to the location on the IdM server where you want to retrieve the **new-private-key-to-an-externally-signed-certificate.pem** secret, for example the current working directory.
 - Set the **action** variable to **member**.
- This is the modified Ansible playbook file for the current example:

```
---
- name: Retrieve data from vault
  hosts: ipaserver
  become: no
  gather_facts: false

  tasks:
    - name: Retrieve data from the service vault
      ipavault:
        ipaadmin_password: Secret123
        name: secret_vault
        service: HTTP/webserver1.idm.example.com
        vault_type: asymmetric
        private_key: "{{ lookup('file', 'service-private.pem') | b64encode }}"
        out: new-private-key-to-an-externally-signed-certificate.pem
        state: retrieved
```

9. Add a section to the playbook that retrieves the data file from the IdM server to the Ansible controller:

```
---
- name: Retrieve data from vault
  hosts: ipaserver
  become: yes
  gather_facts: false
  tasks:
    [...]
    - name: Retrieve data file
      fetch:
        src: new-private-key-to-an-externally-signed-certificate.pem
        dest: ./
        flat: yes
        mode: 0600
```

10. Add a section to the playbook that transfers the retrieved **new-private-key-to-an-externally-signed-certificate.pem** file from the Ansible controller on to the web servers listed in the **web servers** section of the inventory file:

```
---  
- name: Send data file to webserver  
  become: yes  
  gather_facts: no  
  hosts: webserver  
  tasks:  
    - name: Send data to webserver  
      copy:  
        src: new-private-key-to-an-externally-signed-certificate.pem  
        dest: /etc/pki/tls/private/httpd.key  
        mode: 0444
```

11. Save the file.

12. Run the playbook:

```
$ ansible-playbook -v -i inventory.file retrieve-data-asymmetric-vault-copy.yml
```

Additional resources

- For more information about using Ansible to manage IdM vaults and service secrets and about playbook variables, see the README-vault.md Markdown file available in the **/usr/share/doc/ansible-freeipa/** directory and the sample playbooks available in the **/usr/share/doc/ansible-freeipa/playbooks/vault/** directory.

CHAPTER 23. ENSURING THE PRESENCE AND ABSENCE OF SERVICES IN IDM USING ANSIBLE

With the Ansible **service** module, Identity Management (IdM) administrator can ensure that specific services that are not native to IdM are present or absent in IdM. For example, you can use the **service** module to:

- Check that a manually installed service is present on an IdM client and automatically install that service if it is absent. For details, see:
 - [Ensuring the presence of an HTTP service in IdM on an IdM client.](#)
 - [Ensuring the presence of an HTTP service in IdM on a non-IdM client.](#)
 - [Ensuring the presence of an HTTP service on an IdM client without DNS.](#)
- Check that a service enrolled in IdM has a certificate attached and automatically install that certificate if it is absent. For details, see:
 - [Ensuring the presence of an externally-signed certificate in an IdM service entry.](#)
- Allow IdM users and hosts to retrieve and create the service keytab. For details, see:
 - [Allowing IdM users, groups, hosts, or host groups to create a keytab of a service.](#)
 - [Allowing IdM users, groups, hosts, or host groups to retrieve a keytab of a service.](#)
- Allow IdM users and hosts to add a Kerberos alias to a service. For details, see:
 - [Ensuring the presence of a Kerberos principal alias for a service.](#)
- Check that a service is not present on an IdM client and automatically remove that service if it is present. For details, see:
 - [Ensuring the absence of an HTTP service in IdM on an IdM client.](#)

23.1. ENSURING THE PRESENCE OF AN HTTP SERVICE IN IDM USING AN ANSIBLE PLAYBOOK

This section describes how to ensure the presence of an HTTP server in IdM using an Ansible playbook.

Prerequisites

- The system to host the HTTP service is an IdM client.
- You have the IdM administrator password.

Procedure

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

```
$ touch inventory.file
```

2. Open the **inventory.file** and define the IdM server that you want to configure in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:


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

3. Make a copy of the `/usr/share/doc/ansible-freeipa/playbooks/service/service-is-present.yml` Ansible playbook file. For example:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/service/service-is-present.yml
   /usr/share/doc/ansible-freeipa/playbooks/service/service-is-present-copy.yml
```

4. Open the `/usr/share/doc/ansible-freeipa/playbooks/service/service-is-present-copy.yml` Ansible playbook file for editing:

```
---
- name: Playbook to manage IPA service.
  hosts: ipaserver
  become: true
  gather_facts: false

  tasks:
    # Ensure service is present
    - ipaservice:
      ipaadmin_password: Secret123
      name: HTTP/client.idm.example.com
```

5. Adapt the file:
 - Change the IdM administrator password defined by the `ipaadmin_password` variable.
 - Change the name of your IdM client on which the HTTP service is running, as defined by the `name` variable of the `ipaservice` task.
6. Save and exit the file.
7. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
   /usr/share/doc/ansible-freeipa/playbooks/service/service-is-present-copy.yml
```

Verification steps

1. Log into the IdM Web UI as IdM administrator.
2. Navigate to **Identity** → **Services**.

If `HTTP/client.idm.example.com@IDM.EXAMPLE.COM` is listed in the **Services** list, the Ansible playbook has been successfully added to IdM.

Additional resources

- You can secure the communication between the HTTP server and browser clients by [adding TLS encryption to an Apache HTTP Server](#).

- You can request a certificate for the HTTP service from an IdM certificate authority. For more information, see the procedure described in [Obtaining an IdM certificate for a service using certmonger](#).

23.2. ENSURING THE PRESENCE OF AN HTTP SERVICE IN IDM ON A NON-IDM CLIENT USING AN ANSIBLE PLAYBOOK

This section describes how to ensure the presence of an HTTP server in IdM on a host that is not an IdM client using an Ansible playbook. By adding the HTTP server to IdM you are also adding the host to IdM.

Prerequisites

- You have [installed an HTTP service](#) on your host.
- The host on which you have set up HTTP is not an IdM client. Otherwise, follow the steps in [Ensuring the presence of an HTTP service in IdM using an Ansible playbook](#).
- You have the IdM administrator password.
- The DNS A record - or the AAAA record if IPv6 is used - for the host is available.

Procedure

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

```
$ touch inventory.file
```

2. Open the **inventory.file** and define the IdM server that you want to configure in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

3. Make a copy of the **/usr/share/doc/ansible-freeipa/playbooks/service/service-is-present-without-host-check.yml** Ansible playbook file. For example:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/service/service-is-present-without-host-check.yml /usr/share/doc/ansible-freeipa/playbooks/service/service-is-present-without-host-check-copy.yml
```

4. Open the copied file, **/usr/share/doc/ansible-freeipa/playbooks/service/service-is-present-without-host-check-copy.yml**, for editing. Locate the **ipaadmin_password** and **name** variables in the **ipaservice** task:

```
---
- name: Playbook to manage IPA service.
  hosts: ipaserver
  become: true
  gather_facts: false

  tasks:
    # Ensure service is present
    - ipaservice:
```

```
ipaadmin_password: MyPassword123
name: HTTP/www2.example.com
skip_host_check: yes
```

5. Adapt the file:

- Set the **ipaadmin_password** variable to your IdM administrator password.
- Set the **name** variable to the name of the host on which the HTTP service is running.

6. Save and exit the file.

7. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
/usr/share/doc/ansible-freeipa/playbooks/service/service-is-present-without-host-
check-copy.yml
```

Verification steps

1. Log into the IdM Web UI as IdM administrator.
2. Navigate to **Identity** → **Services**.

You can now see **HTTP/client.idm.example.com@IDM.EXAMPLE.COM** listed in the **Services** list.

Additional resources

- You can secure the communication between the HTTP server and the browser clients by [adding TLS encryption to an Apache HTTP Server](#).

23.3. ENSURING THE PRESENCE OF AN HTTP SERVICE ON AN IDM CLIENT WITHOUT DNS USING AN ANSIBLE PLAYBOOK

This section describes how to ensure the presence of an HTTP server running on an IdM client that has no DNS entry using an Ansible playbook. The scenario implied is that the IdM host has no DNS A entry available – or no DNS AAAA entry if IPv6 is used instead of IPv4.

Prerequisites

- The system to host the HTTP service is enrolled in IdM.
- The DNS A or DNS AAAA record for the host may not exist. Otherwise, if the DNS record for the host does exist, follow the procedure in [Ensuring the presence of an HTTP service in IdM using an Ansible playbook](#).
- You have the IdM administrator password.

Procedure

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

```
$ touch inventory.file
```

2. Open the **inventory.file** and define the IdM server that you want to configure in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

3. Make a copy of the **/usr/share/doc/ansible-freeipa/playbooks/service/service-is-present-with-host-force.yml** Ansible playbook file. For example:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/service/service-is-present-with-host-force.yml /usr/share/doc/ansible-freeipa/playbooks/service/service-is-present-with-host-force-copy.yml
```

4. Open the copied file, **/usr/share/doc/ansible-freeipa/playbooks/service/service-is-present-with-host-force-copy.yml**, for editing. Locate the **ipaadmin_password** and **name** variables in the **ipaservice** task:

```
---
- name: Playbook to manage IPA service.
  hosts: ipaserver
  become: true
  gather_facts: false

  tasks:
    # Ensure service is present
    - ipaservice:
      ipaadmin_password: MyPassword123
      name: HTTP/ihavenodns.info
      force: yes
```

5. Adapt the file:
 - Set the **ipaadmin_password** variable to your IdM administrator password.
 - Set the **name** variable to the name of the host on which the HTTP service is running.
6. Save and exit the file.
7. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file /usr/share/doc/ansible-freeipa/playbooks/service/service-is-present-with-host-force-copy.yml
```

Verification steps

1. Log into the IdM Web UI as IdM administrator.
2. Navigate to **Identity → Services**.

You can now see **HTTP/client.idm.example.com@IDM.EXAMPLE.COM** listed in the **Services** list.

Additional resources

- You can secure the communication between the Apache HTTP server and browser clients by [adding TLS encryption to the Apache HTTP Server](#).

23.4. ENSURING THE PRESENCE OF AN EXTERNALLY SIGNED CERTIFICATE IN AN IDM SERVICE ENTRY USING AN ANSIBLE PLAYBOOK

This section describes how to use the **ansible-freeipa service** module to ensure that a certificate issued by an external certificate authority (CA) is attached to the IdM entry of the HTTP service. Having the certificate of an HTTP service signed by an external CA rather than the IdM CA is particularly useful if your IdM CA uses a self-signed certificate.

Prerequisites

- You have [installed an HTTP service](#) on your host.
- You have [enrolled the HTTP service into IdM](#).
- You have the IdM administrator password.
- You have an externally signed certificate whose Subject corresponds to the principal of the HTTP service.

Procedure

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

```
$ touch inventory.file
```

2. Open the **inventory.file** and define the IdM server that you want to configure in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

3. Make a copy of the **/usr/share/doc/ansible-freeipa/playbooks/service/service-member-certificate-present.yml** file, for example:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/service/service-member-certificate-present.yml /usr/share/doc/ansible-freeipa/playbooks/service/service-member-certificate-present-copy.yml
```

4. Optional: If the certificate is in the Privacy Enhanced Mail (PEM) format, convert the certificate to the Distinguished Encoding Rules (DER) format for easier handling through the command-line interface (CLI):

```
$ openssl x509 -outform der -in cert1.pem -out cert1.der
```

5. Decode the **DER** file to standard output using the **base64** command. Use the **-w0** option to disable wrapping:

```
$ base64 cert1.der -w0
MIIC/zCCAeegAwIBAgIUUV74O+4kXeg21o4vxfRRtyJm...
```

6. Copy the certificate from the standard output to the clipboard.
7. Open the `/usr/share/doc/ansible-freeipa/playbooks/service/service-member-certificate-present-copy.yml` file for editing and view its contents:

```
---
- name: Service certificate present.
  hosts: ipaserver
  become: true
  gather_facts: false

  tasks:
    # Ensure service certificate is present
    - ipaservice:
        ipaadmin_password: MyPassword123
        name: HTTP/www.example.com
        certificate: |
          - MIICBjCCAW8CFHnm32VcXaUDGfEGdDL/...
          [...]
        action: member
        state: present
```

8. Adapt the file:
 - Replace the certificate, defined using the **certificate** variable, with the certificate you copied from the CLI. Note that if you use the **certificate:** variable with the "|" pipe character as indicated, you can enter the certificate THIS WAY rather than having it to enter it in a single line. This makes reading the certificate easier.
 - Change the IdM administrator password, defined by the **ipaadmin_password** variable.
 - Change the name of your IdM client on which the HTTP service is running, defined by the **name** variable.
 - Change any other relevant variables.
9. Save and exit the file.
10. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
/usr/share/doc/ansible-freeipa/playbooks/service/service-member-certificate-present-copy.yml
```

Verification steps

1. Log into the IdM Web UI as IdM administrator.
2. Navigate to **Identity** → **Services**.
3. Click the name of the service with the newly added certificate, for example **HTTP/client.idm.example.com**.

In the **Service Certificate** section on the right, you can now see the newly added certificate.

23.5. USING AN ANSIBLE PLAYBOOK TO ALLOW IDM USERS, GROUPS, HOSTS, OR HOST GROUPS TO CREATE A KEYTAB OF A SERVICE

A keytab is a file containing pairs of Kerberos principals and encrypted keys. Keytab files are commonly used to allow scripts to automatically authenticate using Kerberos, without requiring human interaction or access to password stored in a plain-text file. The script is then able to use the acquired credentials to access files stored on a remote system.

As an Identity Management (IdM) administrator, you can allow other users to retrieve or even create a keytab for a service running in IdM. By allowing specific users and user groups to create keytabs, you can delegate the administration of the service to them without sharing the IdM administrator password. This delegation provides a more fine-grained system administration.

This section describes how you can allow specific IdM users, user groups, hosts, and host groups to create a keytab for the HTTP service running on an IdM client. Specifically, it describes how you can allow the **user01** IdM user to create a keytab for the HTTP service running on an IdM client named **client.idm.example.com**.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible controller.
- You have [enrolled the HTTP service into IdM](#).
- The system to host the HTTP service is an IdM client.
- The IdM users and user groups that you want to allow to create the keytab exist in IdM.
- The IdM hosts and host groups that you want to allow to create the keytab exist in IdM.

Procedure

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

```
$ touch inventory.file
```

2. Open the **inventory.file** and define the IdM server that you want to configure in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

3. Make a copy of the **/usr/share/doc/ansible-freeipa/playbooks/service/service-member-allow_create_keytab-present.yml** Ansible playbook file. For example:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/service/service-member-allow_create_keytab-present.yml /usr/share/doc/ansible-freeipa/playbooks/service/service-member-allow_create_keytab-present-copy.yml
```

4. Open the **/usr/share/doc/ansible-freeipa/playbooks/service/service-member-allow_create_keytab-present-copy.yml** Ansible playbook file for editing.

5. Adapt the file by changing the following:

- The IdM administrator password specified by the **ipaadmin_password** variable.
- The name of your IdM client on which the HTTP service is running. In the current example, it is **HTTP/client.idm.example.com**
- The names of IdM users that are listed in the **allow_create_keytab_user:** section. In the current example, it is **user01**.
- The names of IdM user groups that are listed in the **allow_create_keytab_group:** section.
- The names of IdM hosts that are listed in the **allow_create_keytab_host:** section.
- The names of IdM host groups that are listed in the **allow_create_keytab_hostgroup:** section.
- The name of the task specified by the **name** variable in the **tasks** section. After being adapted for the current example, the copied file looks like this:

```
---
- name: Service member allow_create_keytab present
  hosts: ipaserver
  become: true

  tasks:
    - name: Service HTTP/client.idm.example.com members allow_create_keytab present for
      user01
      ipaservice:
        ipaadmin_password: Secret123
        name: HTTP/client.idm.example.com
        allow_create_keytab_user:
          - user01
        action: member
```

6. Save the file.

7. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file
/usr/share/doc/ansible-freeipa/playbooks/service/service-member-
allow_create_keytab-present-copy.yml
```

Verification steps

1. SSH to an IdM server as an IdM user that has the privilege to create a keytab for the particular HTTP service:

```
$ ssh user01@server.idm.example.com
Password:
```

2. Use the **ipa-getkeytab** command to generate the new keytab for the HTTP service:

```
$ ipa-getkeytab -s server.idm.example.com -p HTTP/client.idm.example.com -k
/etc/httpd/conf/krb5.keytab
```


The **-s** option specifies a Key Distribution Center (KDC) server to generate the keytab.

The **-p** option specifies the principal whose keytab you want to create.

The **-k** option specifies the keytab file to append the new key to. The file will be created if it does not exist.

If the command does not result in an error, you have successfully created a keytab of **HTTP/client.idm.example.com** as **user01**.

23.6. USING AN ANSIBLE PLAYBOOK TO ALLOW IDM USERS, GROUPS, HOSTS, OR HOST GROUPS TO RETRIEVE A KEYTAB OF A SERVICE

A keytab is a file containing pairs of Kerberos principals and encrypted keys. Keytab files are commonly used to allow scripts to automatically authenticate using Kerberos, without requiring human interaction or access to a password stored in a plain-text file. The script is then able to use the acquired credentials to access files stored on a remote system.

As IdM administrator, you can allow other users to retrieve or even create a keytab for a service running in IdM.

This section describes how you can allow specific IdM users, user groups, hosts, and host groups to retrieve a keytab for the HTTP service running on an IdM client. Specifically, it describes how to allow the **user01** IdM user to retrieve the keytab of the HTTP service running on **client.idm.example.com**.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible controller.
- You have [enrolled the HTTP service into IdM](#).
- The IdM users and user groups that you want to allow to retrieve the keytab exist in IdM.
- The IdM hosts and host groups that you want to allow to retrieve the keytab exist in IdM.

Procedure

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

```
$ touch inventory.file
```

2. Open the **inventory.file** and define the IdM server that you want to configure in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

3. Make a copy of the **/usr/share/doc/ansible-freeipa/playbooks/service/service-member-allow_retrieve_keytab-present.yml** Ansible playbook file. For example:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/service/service-member-allow_retrieve_keytab-present.yml /usr/share/doc/ansible-freeipa/playbooks/service/service-member-allow_retrieve_keytab-present-copy.yml
```

4. Open the copied file, `/usr/share/doc/ansible-freeipa/playbooks/service/service-member-allow_retrieve_keytab-present-copy.yml`, for editing:

5. Adapt the file:

- Set the **ipaadmin_password** variable to your IdM administrator password.
- Set the **name** variable of the **ipaservice** task to the principal of the HTTP service. In the current example, it is **HTTP/client.idm.example.com**
- Specify the names of IdM users in the **allow_retrieve_keytab_group:** section. In the current example, it is **user01**.
- Specify the names of IdM user groups in the **allow_retrieve_keytab_group:** section.
- Specify the names of IdM hosts in the **allow_retrieve_keytab_group:** section.
- Specify the names of IdM host groups in the **allow_retrieve_keytab_group:** section.
- Specify the name of the task using the **name** variable in the **tasks** section. After being adapted for the current example, the copied file looks like this:

```
---
- name: Service member allow_retrieve_keytab present
  hosts: ipaserver
  become: true

  tasks:
    - name: Service HTTP/client.idm.example.com members allow_retrieve_keytab present for user01
      ipaservice:
        ipaadmin_password: Secret123
        name: HTTP/client.idm.example.com
        allow_retrieve_keytab_user:
          - user01
        action: member
```

6. Save the file.

7. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file /usr/share/doc/ansible-freeipa/playbooks/service/service-member-allow_retrieve_keytab-present-copy.yml
```

Verification steps

1. SSH to an IdM server as an IdM user with the privilege to retrieve a keytab for the HTTP service:

```
$ ssh user01@server.idm.example.com
Password:
```

2. Use the **ipa-getkeytab** command with the **-r** option to retrieve the keytab:

```
$ ipa-getkeytab -r -s server.idm.example.com -p HTTP/client.idm.example.com -k
/etc/httpd/conf/krb5.keytab
```

The **-s** option specifies a Key Distribution Center (KDC) server from which you want to retrieve the keytab.

The **-p** option specifies the principal whose keytab you want to retrieve.

The **-k** option specifies the keytab file to which you want to append the retrieved key. The file will be created if it does not exist.

If the command does not result in an error, you have successfully retrieved a keytab of **HTTP/client.idm.example.com** as **user01**.

23.7. ENSURING THE PRESENCE OF A KERBEROS PRINCIPAL ALIAS OF A SERVICE USING AN ANSIBLE PLAYBOOK

In some scenarios, it is beneficial for IdM administrator to enable IdM users, hosts, or services to authenticate against Kerberos applications using a Kerberos principal alias. These scenarios include:

- The user name changed, but the user should be able to log into the system using both the previous and new user names.
- The user needs to log in using the email address even if the IdM Kerberos realm differs from the email domain.

This section describes how to create the principal alias of **HTTP/mycompany.idm.example.com** for the HTTP service running on **client.idm.example.com**.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible controller.
- You have [set up an HTTP service](#) on your host.
- You have [enrolled the HTTP service into IdM](#).
- The host on which you have set up HTTP is an IdM client.

Procedure

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

```
$ touch inventory.file
```

2. Open the **inventory.file** and define the IdM server that you want to configure in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

3. Make a copy of the `/usr/share/doc/ansible-freeipa/playbooks/service/service-member-principal-present.yml` Ansible playbook file. For example:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/service/service-member-principal-present.yml /usr/share/doc/ansible-freeipa/playbooks/service/service-member-principal-present-copy.yml
```

4. Open the `/usr/share/doc/ansible-freeipa/playbooks/service/service-member-principal-present-copy.yml` Ansible playbook file for editing.
5. Adapt the file by changing the following:
 - The IdM administrator password specified by the `ipaadmin_password` variable.
 - The name of the service specified by the `name` variable. This is the canonical principal name of the service. In the current example, it is `HTTP/client.idm.example.com`.
 - The Kerberos principal alias specified by the `principal` variable. This is the alias you want to add to the service defined by the `name` variable. In the current example, it is `host/mycompany.idm.example.com`.
 - The name of the task specified by the `name` variable in the `tasks` section. After being adapted for the current example, the copied file looks like this:

```
---
- name: Service member principal present
  hosts: ipaserver
  become: true

  tasks:
    - name: Service HTTP/client.idm.example.com member principals
      host/mycompany.idm.exmaple.com present
      ipaservice:
        ipaadmin_password: Secret123
        name: HTTP/client.idm.example.com
        principal:
          - host/mycompany.idm.example.com
      action: member
```

6. Save the file.
7. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file /usr/share/doc/ansible-freeipa/playbooks/service/service-member-principal-present-copy.yml
```

If running the playbook results in 0 unreachable and 0 failed tasks, you have successfully created the `host/mycompany.idm.example.com` Kerberos principal for the `HTTP/client.idm.example.com` service.

Additional resources

- For more information on Kerberos principal aliases and managing them without Ansible, see [Managing Kerberos principal aliases for users, hosts, and services](#).

23.8. ENSURING THE ABSENCE OF AN HTTP SERVICE IN IDM USING AN ANSIBLE PLAYBOOK

This section describes how to unenroll a service from IdM. More specifically, it describes how to use an Ansible playbook to ensure the absence of an HTTP server named **HTTP/client.idm.example.com** in IdM.

Prerequisites

- You have the IdM administrator password.

Procedure

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

```
$ touch inventory.file
```

2. Open the **inventory.file** and define the IdM server that you want to configure in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

3. Make a copy of the **/usr/share/doc/ansible-freeipa/playbooks/service/service-is-absent.yml** Ansible playbook file. For example:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/service/service-is-absent.yml
   /usr/share/doc/ansible-freeipa/playbooks/service/service-is-absent-copy.yml
```

4. Open the **/usr/share/doc/ansible-freeipa/playbooks/service/service-is-absent-copy.yml** Ansible playbook file for editing.

5. Adapt the file by changing the following:

- The IdM administrator password defined by the **ipaadmin_password** variable.
- The Kerberos principal of the HTTP service, as defined by the **name** variable of the **ipaservice** task.

After being adapted for the current example, the copied file looks like this:

```
---
- name: Playbook to manage IPA service.
  hosts: ipaserver
  become: true
  gather_facts: false

  tasks:
    # Ensure service is absent
    - ipaservice:
      ipaadmin_password: Secret123
      name: HTTP/client.idm.example.com
      state: absent
```

6. Save and exit the file.

7. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i path_to_inventory_directory/inventory.file  
/usr/share/doc/ansible-freeipa/playbooks/service/service-is-absent-copy.yml
```

Verification steps

1. Log into the IdM Web UI as IdM administrator.
2. Navigate to **Identity** → **Services**.

If you cannot see the **HTTP/client.idm.example.com@IDM.EXAMPLE.COM** service in the **Services** list, you have successfully ensured its absence in IdM.

Additional resources

- You can see sample Ansible playbooks for ensuring the presence and absence of services in IdM including a list of possible variables in the **README-service.md** Markdown file available in the **/usr/share/doc/ansible-freeipa/** directory.
- You can see sample Ansible playbooks for ensuring the presence and absence of services in IdM in the **/usr/share/doc/ansible-freeipa/playbooks/config** directory.

CHAPTER 24. MANAGING GLOBAL DNS CONFIGURATION IN IDM USING ANSIBLE PLAYBOOKS

Using the Red Hat Ansible Engine **dnsconfig** module, you can configure global configuration for Identity Management (IdM) DNS. Settings defined in global DNS configuration are applied to all IdM DNS servers. However, the global configuration has lower priority than the configuration for a specific IdM DNS zone.

The **dnsconfig** module supports the following variables:

- The global forwarders, specifically their IP addresses and the port used for communication.
- The global forwarding policy: only, first, or none. For more details on these types of DNS forward policies, see [DNS forward policies in IdM](#).
- The synchronization of forward lookup and reverse lookup zones.

Prerequisites

- DNS service is installed on the IdM server. For more information about how to install an IdM server with integrated DNS, see one of the following links:
 - [Installing an IdM server: With integrated DNS, with an integrated CA as the root CA](#)
 - [Installing an IdM server: With integrated DNS, with an external CA as the root CA](#)
 - [Installing an IdM server: With integrated DNS, without a CA](#)

This chapter includes the following sections:

- [How IdM ensures that global forwarders from /etc/resolv.conf are not removed by NetworkManager](#)
- [Ensuring the presence of a DNS global forwarder in IdM using Ansible](#)
- [Ensuring the absence of a DNS global forwarder in IdM using Ansible](#)
- An introduction to [DNS forward policies in IdM](#)
- [Using an Ansible playbook to ensure that the forward first policy is set in IdM DNS global configuration](#)
- [Using an Ansible playbook to ensure that global forwarders are disabled in IdM DNS](#)
- [Using an Ansible playbook to ensure that synchronization of forward and reverse lookup zones is disabled in IdM DNS](#)

24.1. HOW IDM ENSURES THAT GLOBAL FORWARDERS FROM /ETC/RESOLV.CONF ARE NOT REMOVED BY NETWORKMANAGER

Installing Identity Management (IdM) with integrated DNS configures the **/etc/resolv.conf** file to point to the **127.0.0.1** localhost address:

```
# Generated by NetworkManager
search idm.example.com
nameserver 127.0.0.1
```

In certain environments, such as networks that use **Dynamic Host Configuration Protocol** (DHCP), the **NetworkManager** service may revert changes to the `/etc/resolv.conf` file. To make the DNS configuration persistent, the IdM DNS installation process also configures the **NetworkManager** service in the following way:

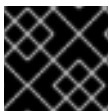
1. The DNS installation script creates an `/etc/NetworkManager/conf.d/zzz-ipa.conf` **NetworkManager** configuration file to control the search order and DNS server list:

```
# auto-generated by IPA installer
[main]
dns=default

[global-dns]
searches=$DOMAIN

[global-dns-domain-*]
servers=127.0.0.1
```

2. The **NetworkManager** service is reloaded, which always creates the `/etc/resolv.conf` file with the settings from the last file in the `/etc/NetworkManager/conf.d/` directory. This is in this case the `zzz-ipa.conf` file.



IMPORTANT

Do not modify the `/etc/resolv.conf` file manually.

24.2. ENSURING THE PRESENCE OF A DNS GLOBAL FORWARDER IN IDM USING ANSIBLE

This section describes how an Identity Management (IdM) administrator can use an Ansible playbook to ensure the presence of a DNS global forwarder in IdM. In the example procedure below, the IdM administrator ensures the presence of a DNS global forwarder to a DNS server with an Internet Protocol (IP) v4 address of **7.7.9.9** and IP v6 address of **2001:db8::1:0** on port **53**.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller. This is the host on which you execute the steps in the procedure.
- You know the IdM administrator password.

Procedure

1. Navigate to the `/usr/share/doc/ansible-freeipa/playbooks/dnsconfig` directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/dnsconfig
```


2. Open your inventory file and make sure that the IdM server that you want to configure is listed in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

3. Make a copy of the **forwarders-absent.yml** Ansible playbook file. For example:

```
$ cp forwarders-absent.yml ensure-presence-of-a-global-forwarder.yml
```

4. Open the **ensure-presence-of-a-global-forwarder.yml** file for editing.

5. Adapt the file by setting the following variables:

- a. Change the **name** variable for the playbook to **Playbook to ensure the presence of a global forwarder in IdM DNS**.
 - b. In the **tasks** section, change the **name** of the task to **Ensure the presence of a DNS global forwarder to 7.7.9.9 and 2001:db8::1:0 on port 53**.
 - c. In the **forwarders** section of the **ipadnsconfig** portion:
 - i. Change the first **ip_address** value to the IPv4 address of the global forwarder: **7.7.9.9**.
 - ii. Change the second **ip_address** value to the IPv6 address of the global forwarder: **2001:db8::1:0**.
 - iii. Verify the **port** value is set to **53**.
 - d. Change the **state** to **present**.
- This the modified Ansible playbook file for the current example:

```
---
- name: Playbook to ensure the presence of a global forwarder in IdM DNS
  hosts: ipaserver
  become: true

  tasks:
    - name: Ensure the presence of a DNS global forwarder to 7.7.9.9 and 2001:db8::1:0 on port
      53
      ipadnsconfig:
        forwarders:
          - ip_address: 7.7.9.9
          - ip_address: 2001:db8::1:0
          port: 53
          state: present
```

6. Save the file.

7. Run the playbook:

```
$ ansible-playbook -v -i inventory.file ensure-presence-of-a-global-forwarder.yml
```

- You can see more sample Ansible playbooks for the **ansible-freeipa ipadnsconfig** module in the **README-dnsconfig.md** Markdown file available in the `/usr/share/doc/ansible-freeipa/` directory. The file also contains the definitions of **ipadnsconfig** variables.

24.3. ENSURING THE ABSENCE OF A DNS GLOBAL FORWARDER IN IDM USING ANSIBLE

This section describes how an Identity Management (IdM) administrator can use an Ansible playbook to ensure the absence of a DNS global forwarder in IdM. In the example procedure below, the IdM administrator ensures the absence of a DNS global forwarder to a DNS server with an Internet Protocol (IP) v4 address of **8.8.6.6** and IP v6 address of **2001:4860:4860::8800** on port **53**.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller. This is the host on which you execute the steps in the procedure.
- You know the IdM administrator password.

Procedure

1. Navigate to the `/usr/share/doc/ansible-freeipa/playbooks/dnsconfig` directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/dnsconfig
```

2. Open your inventory file and make sure that the IdM server that you want to configure is listed in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

3. Make a copy of the **forwarders-absent.yml** Ansible playbook file. For example:

```
$ cp forwarders-absent.yml ensure-absence-of-a-global-forwarder.yml
```

4. Open the **ensure-absence-of-a-global-forwarder.yml** file for editing.
5. Adapt the file by setting the following variables:
 - a. Change the **name** variable for the playbook to **Playbook to ensure the absence of a global forwarder in IdM DNS**.
 - b. In the **tasks** section, change the **name** of the task to **Ensure the absence of a DNS global forwarder to 8.8.6.6 and 2001:4860:4860::8800 on port 53**.
 - c. In the **forwarders** section of the **ipadnsconfig** portion:
 - i. Change the first **ip_address** value to the IPv4 address of the global forwarder: **8.8.6.6**.
 - ii. Change the second **ip_address** value to the IPv6 address of the global forwarder: **2001:4860:4860::8800**.
 - iii. Verify the **port** value is set to **53**.

- d. Verify the **state** is set to **absent**.

This is the modified Ansible playbook file for the current example:

```
---
- name: Playbook to ensure the absence of a global forwarder in IdM DNS
  hosts: ipaserver
  become: true

  tasks:
    - name: Ensure the absence of a DNS global forwarder to 8.8.6.6 and
      2001:4860:4860::8800 on port 53
      ipadnsconfig:
        forwarders:
          - ip_address: 8.8.6.6
          - ip_address: 2001:4860:4860::8800
        port: 53
        state: absent
```

6. Save the file.

7. Run the playbook:

```
$ ansible-playbook -v -i inventory.file ensure-absence-of-a-global-forwarder.yml
```

Additional resources

- You can see more sample Ansible playbooks for the **ansible-freeipa ipadnsconfig** module in the **README-dnsconfig.md** Markdown file available in the `/usr/share/doc/ansible-freeipa/` directory. The file also contains the definitions of **ipadnsconfig** variables.

24.4. DNS FORWARD POLICIES IN IDM

IdM supports the **first** and **only** standard BIND forward policies, as well as the **none** IdM-specific forward policy.

Forward first (*default*)

The IdM BIND service forwards DNS queries to the configured forwarder. If a query fails because of a server error or timeout, BIND falls back to the recursive resolution using servers on the Internet. The **forward first** policy is the default policy, and it is suitable for optimizing DNS traffic.

Forward only

The IdM BIND service forwards DNS queries to the configured forwarder. If a query fails because of a server error or timeout, BIND returns an error to the client. The **forward only** policy is recommended for environments with split DNS configuration.

None (*forwarding disabled*)

DNS queries are not forwarded with the **none** forwarding policy. Disabling forwarding is only useful as a zone-specific override for global forwarding configuration. This option is the IdM equivalent of specifying an empty list of forwarders in BIND configuration.



NOTE

You cannot use forwarding to combine data in IdM with data from other DNS servers. You can only forward queries for specific subzones of the primary zone in IdM DNS.

By default, the BIND service does not forward queries to another server if the queried DNS name belongs to a zone for which the IdM server is authoritative. In such a situation, if the queried DNS name cannot be found in the IdM database, the **NXDOMAIN** answer is returned. Forwarding is not used.

Example 24.1. Example Scenario

The IdM server is authoritative for the **test.example.** DNS zone. BIND is configured to forward queries to the DNS server with the **192.0.2.254** IP address.

When a client sends a query for the **nonexistent.test.example.** DNS name, BIND detects that the IdM server is authoritative for the **test.example.** zone and does not forward the query to the **192.0.2.254.** server. As a result, the DNS client receives the **NXDomain** error message, informing the user that the queried domain does not exist.

24.5. USING AN ANSIBLE PLAYBOOK TO ENSURE THAT THE FORWARD FIRST POLICY IS SET IN IDM DNS GLOBAL CONFIGURATION

This section describes how an Identity Management (IdM) administrator can use an Ansible playbook to ensure that global forwarding policy in IdM DNS is set to **forward first**.

If you use the **forward first** DNS forwarding policy, DNS queries are forwarded to the configured forwarder. If a query fails because of a server error or timeout, BIND falls back to the recursive resolution using servers on the Internet. The forward first policy is the default policy. It is suitable for traffic optimization.

Prerequisites

- You have installed the **ansible-freeipa** package on the Ansible controller, the host on which you execute the procedure. For more information, see [Installing the ansible-freeipa package](#).
- You know the IdM administrator password.
- Your IdM environment contains an integrated DNS server.

Procedure

1. Navigate to the **/usr/share/doc/ansible-freeipa/playbooks/dnsconfig** directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/dnsconfig
```

2. Open your inventory file and ensure that the IdM server that you want to configure is listed in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

3. Make a copy of the **set-configuration.yml** Ansible playbook file. For example:

```
$ cp set-configuration.yml set-forward-policy-to-first.yml
```

4. Open the **set-forward-policy-to-first.yml** file for editing.
5. Adapt the file by setting the following variables in the **ipadnsconfig** task section:
 - Set the **ipaadmin_password** variable to your IdM administrator password.
 - Set the **forward_policy** variable to **first**.
Delete all the other lines of the original playbook that are irrelevant. This is the modified Ansible playbook file for the current example:

```
---
- name: Playbook to set global forwarding policy to first
  hosts: ipaserver
  become: true

  tasks:
  - name: Set global forwarding policy to first.
    ipadnsconfig:
      ipaadmin_password: Secret123
      forward_policy: first
```

6. Save the file.
7. Run the playbook:

```
$ ansible-playbook -v -i inventory.file set-forward-policy-to-first.yml
```

Additional resources

- For more information on forwarding policy types available in IdM DNS, see [DNS forward policies in IdM](#).
- For more sample Ansible playbooks using the **ansible-freeipa ipadnsconfig** module, see the **README-dnsconfig.md** Markdown file available in the `/usr/share/doc/ansible-freeipa/` directory. The file also contains the definitions of the **ipadnsconfig** variables.
- For more sample Ansible playbooks using the **ipadnsconfig** module, see the `/usr/share/doc/ansible-freeipa/playbooks/dnsconfig` directory.

24.6. USING AN ANSIBLE PLAYBOOK TO ENSURE THAT GLOBAL FORWARDERS ARE DISABLED IN IDM DNS

This section describes how an Identity Management (IdM) administrator can use an Ansible playbook to ensure that global forwarders are disabled in IdM DNS. The disabling is done by setting the **forward_policy** variable to **none**.

Disabling global forwarders causes DNS queries not to be forwarded. Disabling forwarding is only useful as a zone-specific override for global forwarding configuration. This options is the IdM equivalent of specifying an empty list of forwarders in BIND configuration.

Prerequisites

- You have installed the **ansible-freeipa** package on the Ansible controller, the host on which you execute the procedure. For more information, see [Installing the ansible-freeipa package](#).
- You know the IdM administrator password.
- Your IdM environment contains an integrated DNS server.

Procedure

1. Navigate to the **/usr/share/doc/ansible-freeipa/playbooks/dnsconfig** directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/dnsconfig
```

2. Open your inventory file and ensure that the IdM server that you want to configure is listed in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

3. Make a copy of the **disable-global-forwarders.yml** Ansible playbook file. For example:

```
$ cp disable-global-forwarders.yml disable-global-forwarders-copy.yml
```

4. Open the **disable-global-forwarders-copy.yml** file for editing.
5. Adapt the file by setting the following variables in the **ipadnsconfig** task section:
 - Set the **ipaadmin_password** variable to your IdM administrator password.
 - Set the **forward_policy** variable to **none**.This is the modified Ansible playbook file for the current example:

```
---
- name: Playbook to disable global DNS forwarders
  hosts: ipaserver
  become: true

  tasks:
    - name: Disable global forwarders.
      ipadnsconfig:
        ipaadmin_password: Secret123
        forward_policy: none
```

6. Save the file.
7. Run the playbook:

```
$ ansible-playbook -v -i inventory.file disable-global-forwarders-copy.yml
```

Additional resources

- For more information on forwarding policy types available in IdM DNS, see [DNS forward policies in IdM](#).
- For more sample Ansible playbooks using the **ansible-freeipa ipadnsconfig** module, see the **README-dnsconfig.md** Markdown file available in the `/usr/share/doc/ansible-freeipa/` directory. The file also contains the definitions of the **ipadnsconfig** variables.
- For more sample Ansible playbooks using the **ipadnsconfig** module, see the `/usr/share/doc/ansible-freeipa/playbooks/dnsconfig` directory.

24.7. USING AN ANSIBLE PLAYBOOK TO ENSURE THAT SYNCHRONIZATION OF FORWARD AND REVERSE LOOKUP ZONES IS DISABLED IN IDM DNS

This section describes how an Identity Management (IdM) administrator can use an Ansible playbook to ensure that forward and reverse lookup zones are not synchronized in IdM DNS.

Prerequisites

- You have installed the **ansible-freeipa** package on the Ansible controller, the host on which you execute the procedure. For more information, see [Installing the ansible-freeipa package](#).
- You know the IdM administrator password.
- Your IdM environment contains an integrated DNS server.

Procedure

1. Navigate to the `/usr/share/doc/ansible-freeipa/playbooks/dnsconfig` directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/dnsconfig
```

2. Open your inventory file and ensure that the IdM server that you want to configure is listed in the **[ipaserver]** section. For example, to instruct Ansible to configure `server.idm.example.com`, enter:

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

3. Make a copy of the **disallow-reverse-sync.yml** Ansible playbook file. For example:

```
$ cp disallow-reverse-sync.yml disallow-reverse-sync-copy.yml
```

4. Open the **disallow-reverse-sync-copy.yml** file for editing.
5. Adapt the file by setting the following variables in the **ipadnsconfig** task section:
 - Set the **ipaadmin_password** variable to your IdM administrator password.

- Set the **allow_sync_ptr** variable to **no**.
This is the modified Ansible playbook file for the current example:

```
---
- name: Playbook to disallow reverse record synchronization
  hosts: ipaserver
  become: true

  tasks:
    - name: Disallow reverse record synchronization.
      ipadnsconfig:
        ipadmin_password: Secret123
        allow_sync_ptr: no
```

6. Save the file.
7. Run the playbook:

```
$ ansible-playbook -v -i inventory.file disallow-reverse-sync-copy.yml
```

Additional resources

- For more sample Ansible playbooks using the **ansible-freeipa ipadnsconfig** module, see the **README-dnsconfig.md** Markdown file available in the **/usr/share/doc/ansible-freeipa/** directory. The file also contains the definitions of the **ipadnsconfig** variables.
- For more sample Ansible playbooks using the **ipadnsconfig** module, see the **/usr/share/doc/ansible-freeipa/playbooks/dnsconfig** directory.

CHAPTER 25. USING ANSIBLE PLAYBOOKS TO MANAGE IDM DNS ZONES

As Identity Management (IdM) administrator, you can manage how IdM DNS zones work using the **dnszone** module available in the **ansible-freeipa** package. The chapter describes the following topics and procedures:

- [What DNS zone types are supported in IdM](#)
- [What DNS attributes you can configure in IdM](#)
- [How to use an Ansible playbook to create a primary zone in IdM DNS](#)
- [How to use an Ansible playbook to ensure the presence of a primary IdM DNS zone with multiple variables](#)
- [How to use an Ansible playbook to ensure the presence of a zone for reverse DNS lookup when an IP address is given](#)

Prerequisites

- DNS service is installed on the IdM server. For more information about how to use Red Hat Ansible Engine to install an IdM server with integrated DNS, see [Installing an Identity Management server using an Ansible playbook](#).

25.1. SUPPORTED DNS ZONE TYPES

Identity Management (IdM) supports two types of DNS zones: *primary* and *forward* zones. This section describes these two types of zones and includes an example scenario for DNS forwarding.



NOTE

This guide uses the BIND terminology for zone types which is different from the terminology used for Microsoft Windows DNS. Primary zones in BIND serve the same purpose as *forward lookup zones* and *reverse lookup zones* in Microsoft Windows DNS. Forward zones in BIND serve the same purpose as *conditional forwarders* in Microsoft Windows DNS.

Primary DNS zones

Primary DNS zones contain authoritative DNS data and can accept dynamic DNS updates. This behavior is equivalent to the **type master** setting in standard BIND configuration. You can manage primary zones using the **ipa dnszone-*** commands.

In compliance with standard DNS rules, every primary zone must contain **start of authority** (SOA) and **nameserver** (NS) records. IdM generates these records automatically when the DNS zone is created, but you must copy the NS records manually to the parent zone to create proper delegation.

In accordance with standard BIND behavior, queries for names for which the server is not authoritative are forwarded to other DNS servers. These DNS servers, so called forwarders, may or may not be authoritative for the query.

Example 25.1. Example scenario for DNS forwarding

The IdM server contains the **test.example.** primary zone. This zone contains an NS delegation record for the **sub.test.example.** name. In addition, the **test.example.** zone is configured with the **192.0.2.254** forwarder IP address for the **sub.test.example** subzone.

A client querying the name **nonexistent.test.example.** receives the **NXDomain** answer, and no forwarding occurs because the IdM server is authoritative for this name.

On the other hand, querying for the **host1.sub.test.example.** name is forwarded to the configured forwarder **192.0.2.254** because the IdM server is not authoritative for this name.

Forward DNS zones

From the perspective of IdM, forward DNS zones do not contain any authoritative data. In fact, a forward "zone" usually only contains two pieces of information:

- A domain name
- The IP address of a DNS server associated with the domain

All queries for names belonging to the domain defined are forwarded to the specified IP address. This behavior is equivalent to the **type forward** setting in standard BIND configuration. You can manage forward zones using the **ipa dnsforwardzone-*** commands.

Forward DNS zones are especially useful in the context of IdM-Active Directory (AD) trusts. If the IdM DNS server is authoritative for the **idm.example.com** zone and the AD DNS server is authoritative for the **ad.example.com** zone, then **ad.example.com** is a DNS forward zone for the **idm.example.com** primary zone. That means that when a query comes from an IdM client for the IP address of **somehost.ad.example.com**, the query is forwarded to an AD domain controller specified in the **ad.example.com** IdM DNS forward zone.

25.2. CONFIGURATION ATTRIBUTES OF PRIMARY IDM DNS ZONES

Identity Management (IdM) creates a new zone with certain default configuration, such as the refresh periods, transfer settings, or cache settings. In [IdM DNS zone attributes](#), you can find the attributes of the default zone configuration that you can modify using one of the following options:

- The **dnszone-mod** command in the command-line interface (CLI). For more information, see [Editing the configuration of a primary DNS zone in IdM CLI](#).
- The IdM Web UI. For more information, see [Editing the configuration of a primary DNS zone in IdM Web UI](#).
- An Ansible playbook that uses the **ipadnszone** module. For more information, see [Managing DNS zones in IdM](#).

Along with setting the actual information for the zone, the settings define how the DNS server handles the *start of authority* (SOA) record entries and how it updates its records from the DNS name server.

Table 25.1. IdM DNS zone attributes

Attribute	ansible-freeipa variable	Description
Authoritative name server	name_server	<p>Sets the domain name of the primary DNS name server, also known as SOA MNAME.</p> <p>By default, each IdM server advertises itself in the SOA MNAME field. Consequently, the value stored in LDAP using --name-server is ignored.</p>
Administrator e-mail address	admin_email	Sets the email address to use for the zone administrator. This defaults to the root account on the host.
SOA serial	serial	Sets a serial number in the SOA record. Note that IdM sets the version number automatically and users are not expected to modify it.
SOA refresh	refresh	Sets the interval, in seconds, for a secondary DNS server to wait before requesting updates from the primary DNS server.
SOA retry	retry	Sets the time, in seconds, to wait before retrying a failed refresh operation.
SOA expire	expire	Sets the time, in seconds, that a secondary DNS server will try to perform a refresh update before ending the operation attempt.
SOA minimum	minimum	Sets the time to live (TTL) value in seconds for negative caching according to RFC 2308 .
SOA time to live	ttl	Sets TTL in seconds for records at zone apex. In zone example.com , for instance, all records (A, NS, or SOA) under name example.com are configured, but no other domain names, like test.example.com , are affected.
Default time to live	default_ttl	Sets the default time to live (TTL) value in seconds for negative caching for all values in a zone that never had an individual TTL value set before. Requires a restart of the named-pkcs11 service on all IdM DNS servers after changes to take effect.
BIND update policy	update_policy	Sets the permissions allowed to clients in the DNS zone.
Dynamic update	dynamic_update=TRUE FALSE	<p>Enables dynamic updates to DNS records for clients.</p> <p>Note that if this is set to false, IdM client machines will not be able to add or update their IP address.</p>
Allow transfer	allow_transfer=string	<p>Gives a list of IP addresses or network names which are allowed to transfer the given zone, separated by semicolons (;).</p> <p>Zone transfers are disabled by default. The default allow_transfer value is none.</p>

Attribute	ansible-freeipa variable	Description
Allow query	allow_query	Gives a list of IP addresses or network names which are allowed to issue DNS queries, separated by semicolons (;).
Allow PTR sync	allow_sync_ptr= 1 0	Sets whether A or AAAA records (forward records) for the zone will be automatically synchronized with the PTR (reverse) records.
Zone forwarders	forwarder= <i>IP_address</i>	Specifies a forwarder specifically configured for the DNS zone. This is separate from any global forwarders used in the IdM domain. To specify multiple forwarders, use the option multiple times.
Forward policy	forward_policy= none only first	Specifies the forward policy. For information about the supported policies, see DNS forward policies in IdM .

Additional resources

- You can see more definitions of the attributes of the **ansible-freeipa ipadnszone** module in the **README-dnszone.md** Markdown file available in the `/usr/share/doc/ansible-freeipa/` directory.

25.3. USING ANSIBLE TO CREATE A PRIMARY ZONE IN IDM DNS

This section shows how an Identity Management (IdM) administrator can use an Ansible playbook to ensure that a primary DNS zone exists. In the example used in the procedure below, an IdM administrator ensures the presence of the **zone.idm.example.com** DNS zone.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller. This is the host on which you execute the steps in the procedure.
- You know the IdM administrator password.

Procedure

- Navigate to the `/usr/share/doc/ansible-freeipa/playbooks/dnszone` directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/dnszone
```

- Open your inventory file and ensure that the IdM server that you want to configure is listed in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

3. Make a copy of the **dnszone-present.yml** Ansible playbook file. For example:

```
$ cp dnszone-present.yml dnszone-present-copy.yml
```

4. Open the **dnszone-present-copy.yml** file for editing.
5. Adapt the file by setting the following variables in the **ipadnszone** task section:

- Set the **ipaadmin_password** variable to your IdM administrator password.
- Set the **zone_name** variable to **zone.idm.example.com**.
This is the modified Ansible playbook file for the current example:

```
---
- name: Ensure dnszone present
  hosts: ipaserver
  become: true

  tasks:
    - name: Ensure zone is present.
      ipadnszone:
        ipaadmin_password: Secret123
        zone_name: zone.idm.example.com
        state: present
```

6. Save the file.
7. Run the playbook:

```
$ ansible-playbook -v -i inventory.file dnszone-present-copy.yml
```

Additional resources

- For more information on DNS zone, see [Supported DNS zone types](#).
- You can see more sample Ansible playbooks for the **ansible-freeipa ipadnszone** module in the **README-dnszone.md** Markdown file available in the **/usr/share/doc/ansible-freeipa/** directory. The file also contains the definitions of the **ipadnszone** variables.
- You can see sample Ansible playbooks for the **ipadnszone** module in the **/usr/share/doc/ansible-freeipa/playbooks/dnszone** directory.

25.4. USING AN ANSIBLE PLAYBOOK TO ENSURE THE PRESENCE OF A PRIMARY DNS ZONE IN IDM WITH MULTIPLE VARIABLES

This section shows how an Identity Management (IdM) administrator can use an Ansible playbook to ensure that a primary DNS zone exists. In the example used in the procedure below, an IdM administrator ensures the presence of the **zone.idm.example.com** DNS zone. The Ansible playbook configures multiple parameters of the zone.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller. This is the host on which you execute the steps in the procedure.

- You know the IdM administrator password.

Procedure

1. Navigate to the `/usr/share/doc/ansible-freeipa/playbooks/dnszone` directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/dnszone
```

2. Open your inventory file and ensure that the IdM server that you want to configure is listed in the `[ipaserver]` section. For example, to instruct Ansible to configure `server.idm.example.com`, enter:

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

3. Make a copy of the `dnszone-all-params.yml` Ansible playbook file. For example:

```
$ cp dnszone-all-params.yml dnszone-all-params-copy.yml
```

4. Open the `dnszone-all-params-copy.yml` file for editing.
5. Adapt the file by setting the following variables in the `ipadnszone` task section:
 - Set the `ipaadmin_password` variable to your IdM administrator password.
 - Set the `zone_name` variable to `zone.idm.example.com`.
 - Set the `allow_sync_ptr` variable to true if you want to allow the synchronization of forward and reverse records, that is the synchronization of A and AAAA records with PTR records.
 - Set the `dynamic_update` variable to true to enable IdM client machines to add or update their IP addresses.
 - Set the `dnssec` variable to true to allow inline DNSSEC signing of records in the zone.
 - Set the `allow_transfer` variable to the IP addresses of secondary name servers in the zone.
 - Set the `allow_query` variable to the IP addresses or networks that are allowed to issue queries.
 - Set the `forwarders` variable to the IP addresses of global forwarders.
 - Set the `serial` variable to the SOA record serial number.
 - Define the `refresh`, `retry`, `expire`, `minimum`, `ttl`, and `default_ttl` values for DNS records in the zone.
 - Define the NSEC3PARAM record for the zone using the `nsec3param_rec` variable.
 - Set the `skip_overlap_check` variable to true to force DNS creation even if it overlaps with an existing zone.
 - Set the `skip_nameserver_check` to true to force DNS zone creation even if the nameserver is not resolvable.

This is the modified Ansible playbook file for the current example:

■

```

---
- name: Ensure dnszone present
  hosts: ipaserver
  become: true

  tasks:
  - name: Ensure zone is present.
    ipadnszone:
      ipadmin_password: Secret123
      zone_name: zone.idm.example.com
      allow_sync_ptr: true
      dynamic_update: true
      dnssec: true
      allow_transfer:
        - 1.1.1.1
        - 2.2.2.2
      allow_query:
        - 1.1.1.1
        - 2.2.2.2
      forwarders:
        - ip_address: 8.8.8.8
        - ip_address: 8.8.4.4
      port: 52
      serial: 1234
      refresh: 3600
      retry: 900
      expire: 1209600
      minimum: 3600
      ttl: 60
      default_ttl: 90
      name_server: server.idm.example.com.
      admin_email: admin.admin@idm.example.com
      nsec3param_rec: "1 7 100 0123456789abcdef"
      skip_overlap_check: true
      skip_nameserver_check: true
      state: present

```

6. Save the file.

7. Run the playbook:

```
$ ansible-playbook -v -i inventory.file dnszone-all-params-copy.yml
```

Additional resources

- For more information on DNS zone, see [Supported DNS zone types](#).
- For more information on what DNS zone attributes you can configure in IdM, see [Configuration attributes of primary IdM DNS zones](#).
- You can see more sample Ansible playbooks for the **ansible-freeipa ipadnszone** module in the **README-dnszone.md** Markdown file available in the `/usr/share/doc/ansible-freeipa/` directory. The file also contains the definitions of the **ipadnszone** variables.

- You can see sample Ansible playbooks for the **ipadnszone** module in the **/usr/share/doc/ansible-freeipa/playbooks/dnszone** directory.

25.5. USING AN ANSIBLE PLAYBOOK TO ENSURE THE PRESENCE OF A ZONE FOR REVERSE DNS LOOKUP WHEN AN IP ADDRESS IS GIVEN

This section shows how an Identity Management (IdM) administrator can use an Ansible playbook to ensure that a reverse DNS zone exists. In the example used in the procedure below, an IdM administrator ensures the presence of a reverse DNS lookup zone using the IP address and prefix length of an IdM host.

Providing the prefix length of the IP address of your DNS server using the **name_from_ip** variable allows you to control the zone name. If you do not state the prefix length, the system queries DNS servers for zones and, based on the **name_from_ip** value of **192.168.1.2**, the query can return any of the following DNS zones:

- **1.168.192.in-addr.arpa.**
- **168.192.in-addr.arpa.**
- **192.in-addr.arpa.**

Because the zone returned by the query might not be what you expect, **name_from_ip** can only be used with the **state** option set to **present** to prevent accidental removals of zones.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller. This is the host on which you execute the steps in the procedure.
- You know the IdM administrator password.

Procedure

1. Navigate to the **/usr/share/doc/ansible-freeipa/playbooks/dnszone** directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/dnszone
```

2. Open your inventory file and ensure that the IdM server that you want to configure is listed in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

3. Make a copy of the **dnszone-reverse-from-ip.yml** Ansible playbook file. For example:

```
$ cp dnszone-reverse-from-ip.yml dnszone-reverse-from-ip-copy.yml
```

4. Open the **dnszone-reverse-from-ip-copy.yml** file for editing.
5. Adapt the file by setting the following variables in the **ipadnszone** task section:
 - Set the **ipaadmin_password** variable to your IdM administrator password.

- Set the **name_from_ip** variable to the IP of your IdM nameserver, and provide its prefix length.

This is the modified Ansible playbook file for the current example:

```
---
- name: Ensure dnszone present
  hosts: ipaserver
  become: true

  tasks:
    - name: Ensure zone for reverse DNS lookup is present.
      ipadnszone:
        ipadmin_password: Secret123
        name_from_ip: 192.168.1.2/24
        state: present
        register: result
    - name: Display inferred zone name.
      debug:
        msg: "Zone name: {{ result.dnszone.name }}"
```

The playbook creates a zone for reverse DNS lookup from the **192.168.1.2** IP address and its prefix length of 24. Next, the playbook displays the resulting zone name.

6. Save the file.
7. Run the playbook:

```
$ ansible-playbook -v -i inventory.file dnszone-reverse-from-ip-copy.yml
```

Additional resources

- For more information on DNS zone, see [Supported DNS zone types](#).
- You can see more sample Ansible playbooks for the **ansible-freeipa ipadnszone** module in the **README-dnszone.md** Markdown file available in the **/usr/share/doc/ansible-freeipa/** directory. The file also contains the definitions of the **ipadnszone** variables.
- You can see sample Ansible playbooks for the **ipadnszone** module in the **/usr/share/doc/ansible-freeipa/playbooks/dnszone** directory.

CHAPTER 26. USING ANSIBLE TO MANAGE DNS LOCATIONS IN IDM

As Identity Management (IdM) administrator, you can manage IdM DNS locations using the **location** module available in the **ansible-freeipa** package. The chapter describes the following topics and procedures:

- [DNS-based service discovery](#)
- [Deployment considerations for DNS locations](#)
- [DNS time to live \(TTL\)](#)
- [Using Ansible to ensure an IdM location is present](#)
- [Using Ansible to ensure an IdM location is absent](#)

26.1. DNS-BASED SERVICE DISCOVERY

DNS-based service discovery is a process in which a client uses the DNS protocol to locate servers in a network that offer a specific service, such as **LDAP** or **Kerberos**. One typical type of operation is to allow clients to locate authentication servers within the closest network infrastructure, because they provide a higher throughput and lower network latency, lowering overall costs.

The major advantages of service discovery are:

- No need for clients to be explicitly configured with names of nearby servers.
- DNS servers are used as central providers of policy. Clients using the same DNS server have access to the same policy about service providers and their preferred order.

In an Identity Management (IdM) domain, DNS service records (SRV records) exist for **LDAP**, **Kerberos**, and other services. For example, the following command queries the DNS server for hosts providing a TCP-based **Kerberos** service in an IdM DNS domain:

Example 26.1. DNS location independent results

```
$ dig -t SRV +short _kerberos._tcp.idm.example.com
0 100 88 idmsvr-01.idm.example.com.
0 100 88 idmsvr-02.idm.example.com.
```

The output contains the following information:

- **0** (priority): Priority of the target host. A lower value is preferred.
- **100** (weight). Specifies a relative weight for entries with the same priority. For further information, see [RFC 2782, section 3](#).
- **88** (port number): Port number of the service.
- Canonical name of the host providing the service.

In the example, the two host names returned have the same priority and weight. In this case, the client uses a random entry from the result list.

When the client is, instead, configured to query a DNS server that is configured in a DNS location, the output differs. For IdM servers that are assigned to a location, tailored values are returned. In the example below, the client is configured to query a DNS server in the location **germany**:

Example 26.2. DNS location-based results

```
$ dig -t SRV +short _kerberos._tcp.idm.example.com
_kerberos._tcp.germany._locations.idm.example.com.
0 100 88 idmsvr-01.idm.example.com.
50 100 88 idmsvr-02.idm.example.com.
```

The IdM DNS server automatically returns a DNS alias (CNAME) pointing to a DNS location specific SRV record which prefers local servers. This CNAME record is shown in the first line of the output. In the example, the host **idmsvr-01.idm.example.com** has the lowest priority value and is therefore preferred. The **idmsvr-02.idm.example.com** has a higher priority and thus is used only as backup for cases when the preferred host is unavailable.

26.2. DEPLOYMENT CONSIDERATIONS FOR DNS LOCATIONS

Identity Management (IdM) can generate location-specific service (SRV) records when using the integrated DNS. Because each IdM DNS server generates location-specific SRV records, you have to install at least one IdM DNS server in each DNS location.

The client's affinity to a DNS location is only defined by the DNS records received by the client. For this reason, you can combine IdM DNS servers with non-IdM DNS consumer servers and recursors if the clients doing DNS service discovery resolve location-specific records from IdM DNS servers.

In the majority of deployments with mixed IdM and non-IdM DNS services, DNS recursors select the closest IdM DNS server automatically by using round-trip time metrics. Typically, this ensures that clients using non-IdM DNS servers are getting records for the nearest DNS location and thus use the optimal set of IdM servers.

26.3. DNS TIME TO LIVE (TTL)

Clients can cache DNS resource records for an amount of time that is set in the zone's configuration. Because of this caching, a client might not be able to receive the changes until the time to live (TTL) value expires. The default TTL value in Identity Management (IdM) is **1 day**.

If your client computers roam between sites, you should adapt the TTL value for your IdM DNS zone. Set the value to a lower value than the time clients need to roam between sites. This ensures that cached DNS entries on the client expire before they reconnect to another site and thus query the DNS server to refresh location-specific SRV records.

Additional resources

- For further information how to modify the default TTL of a DNS zone, see [Configuration attributes of primary IdM DNS zones](#).

26.4. USING ANSIBLE TO ENSURE AN IDM LOCATION IS PRESENT

As a system administrator of Identity Management (IdM), you can configure IdM DNS locations to allow clients to locate authentication servers within the closest network infrastructure.

The following procedure describes how to use an Ansible playbook to ensure a DNS location is present in IdM. The example describes how to ensure that the **germany** DNS location is present in IdM. As a result, you can assign particular IdM servers to this location so that local IdM clients can use them to reduce server response time.

Prerequisites

- You know the IdM administrator password.
- You have installed the [ansible-freeipa](#) package on the Ansible control node.
- The example assumes that you have [created and configured](#) the `~/MyPlaybooks/` directory as a central location to store copies of sample playbooks.
- You understand the [deployment considerations for DNS locations](#).

Procedure

1. Navigate to the `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks/
```

2. Make a copy of the **location-present.yml** file located in the `/usr/share/doc/ansible-freeipa/playbooks/location/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/location/location-present.yml location-present-copy.yml
```

3. Open the **location-present-copy.yml** Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the **ipalocation** task section:
 - Adapt the **name** of the task to correspond to your use case.
 - Set the **ipaadmin_password** variable to the password of the IdM administrator.
 - Set the **name** variable to the name of the location.

This is the modified Ansible playbook file for the current example:

```
---
- name: location present example
  hosts: ipaserver
  become: true

  tasks:
  - name: Ensure that the "germany" location is present
    ipalocation:
      ipaadmin_password: Secret123
      name: germany
```

5. Save the file.
6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i inventory location-present-copy.yml
```

Additional resources

- To configure specific servers for the IdM locations that now exist in IdM, see [Assigning an IdM server to a DNS location using the IdM Web UI](#) or [Assigning an IdM server to a DNS location using the IdM CLI](#).

26.5. USING ANSIBLE TO ENSURE AN IDM LOCATION IS ABSENT

As a system administrator of Identity Management (IdM), you can configure IdM DNS locations to allow clients to locate authentication servers within the closest network infrastructure.

The following procedure describes how to use an Ansible playbook to ensure that a DNS location is absent in IdM. The example describes how to ensure that the **germany** DNS location is absent in IdM. As a result, you cannot assign particular IdM servers to this location and local IdM clients cannot use them.

Prerequisites

- You know the IdM administrator password.
- No IdM server is assigned to the **germany** DNS location.
- You have installed the [ansible-freeipa](#) package on the Ansible control node.
- The example assumes that you have [created and configured](#) the `~/MyPlaybooks/` directory as a central location to store copies of sample playbooks.

Procedure

1. Navigate to the `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks/
```

2. Make a copy of the **location-absent.yml** file located in the `/usr/share/doc/ansible-freeipa/playbooks/location/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/location/location-absent.yml location-absent-copy.yml
```

3. Open the **location-absent-copy.yml** Ansible playbook file for editing.
4. Adapt the file by setting the following variables in the **ipalocation** task section:
 - Adapt the **name** of the task to correspond to your use case.
 - Set the **ipaadmin_password** variable to the password of the IdM administrator.
 - Set the **name** variable to the name of the DNS location.
 - Make sure that the **state** variable is set to **absent**.

This is the modified Ansible playbook file for the current example:

```
---
- name: location absent example
  hosts: ipaserver
  become: true

  tasks:
  - name: Ensure that the "germany" location is absent
    ipalocation:
      ipadmin_password: Secret123
      name: germany
      state: absent
```

5. Save the file.
6. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i inventory location-absent-copy.yml
```

26.6. ADDITIONAL RESOURCES

- You can see more sample Ansible playbooks for the **ansible-freeipa ipalocation** module in the **README-location.md** file available in the **/usr/share/doc/ansible-freeipa/** directory. The file also contains the definitions of the **ipalocation** variables.
- You can see sample Ansible playbooks for the **ipalocation** module in the **/usr/share/doc/ansible-freeipa/playbooks/location** directory.

CHAPTER 27. MANAGING DNS FORWARDING IN IDM

The following procedures describe how to configure DNS global forwarders and DNS forward zones in the Identity Management (IdM) Web UI, the IdM CLI, and using Ansible:

- [Section 27.1, “The two roles of an IdM DNS server”](#)
- [Section 27.2, “DNS forward policies in IdM”](#)
- [Section 27.3, “Adding a global forwarder in the IdM Web UI”](#)
- [Section 27.4, “Adding a global forwarder in the CLI”](#)
- [Section 27.5, “Adding a DNS Forward Zone in the IdM Web UI”](#)
- [Section 27.6, “Adding a DNS Forward Zone in the CLI”](#)
- [Section 27.7, “Establishing a DNS Global Forwarder in IdM using Ansible”](#)
- [Section 27.8, “Ensuring the presence of a DNS global forwarder in IdM using Ansible”](#)
- [Section 27.9, “Ensuring the absence of a DNS global forwarder in IdM using Ansible”](#)
- [Section 27.10, “Ensuring DNS Global Forwarders are disabled in IdM using Ansible”](#)
- [Section 27.11, “Ensuring the presence of a DNS Forward Zone in IdM using Ansible”](#)
- [Section 27.12, “Ensuring a DNS Forward Zone has multiple forwarders in IdM using Ansible”](#)
- [Section 27.13, “Ensuring a DNS Forward Zone is disabled in IdM using Ansible”](#)
- [Section 27.14, “Ensuring the absence of a DNS Forward Zone in IdM using Ansible”](#)

27.1. THE TWO ROLES OF AN IDM DNS SERVER

DNS forwarding affects how a DNS service answers DNS queries. By default, the Berkeley Internet Name Domain (BIND) service integrated with IdM acts as both an *authoritative* and a *recursive* DNS server:

Authoritative DNS server

When a DNS client queries a name belonging to a DNS zone for which the IdM server is authoritative, BIND replies with data contained in the configured zone. Authoritative data always takes precedence over any other data.

Recursive DNS server

When a DNS client queries a name for which the IdM server is not authoritative, BIND attempts to resolve the query using other DNS servers. If forwarders are not defined, BIND asks the root servers on the Internet and uses a recursive resolution algorithm to answer the DNS query.

In some cases, it is not desirable to let BIND contact other DNS servers directly and perform the recursion based on data available on the Internet. You can configure BIND to use another DNS server, a *forwarder*, to resolve the query.

When you configure BIND to use a forwarder, queries and answers are forwarded back and forth between the IdM server and the forwarder, and the IdM server acts as the DNS cache for non-authoritative data.

27.2. DNS FORWARD POLICIES IN IDM

IdM supports the **first** and **only** standard BIND forward policies, as well as the **none** IdM-specific forward policy.

Forward first (*default*)

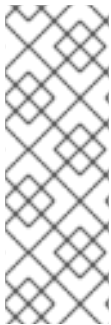
The IdM BIND service forwards DNS queries to the configured forwarder. If a query fails because of a server error or timeout, BIND falls back to the recursive resolution using servers on the Internet. The **forward first** policy is the default policy, and it is suitable for optimizing DNS traffic.

Forward only

The IdM BIND service forwards DNS queries to the configured forwarder. If a query fails because of a server error or timeout, BIND returns an error to the client. The **forward only** policy is recommended for environments with split DNS configuration.

None (*forwarding disabled*)

DNS queries are not forwarded with the **none** forwarding policy. Disabling forwarding is only useful as a zone-specific override for global forwarding configuration. This option is the IdM equivalent of specifying an empty list of forwarders in BIND configuration.



NOTE

You cannot use forwarding to combine data in IdM with data from other DNS servers. You can only forward queries for specific subzones of the primary zone in IdM DNS.

By default, the BIND service does not forward queries to another server if the queried DNS name belongs to a zone for which the IdM server is authoritative. In such a situation, if the queried DNS name cannot be found in the IdM database, the **NXDOMAIN** answer is returned. Forwarding is not used.

Example 27.1. Example Scenario

The IdM server is authoritative for the **test.example.** DNS zone. BIND is configured to forward queries to the DNS server with the **192.0.2.254** IP address.

When a client sends a query for the **nonexistent.test.example.** DNS name, BIND detects that the IdM server is authoritative for the **test.example.** zone and does not forward the query to the **192.0.2.254.** server. As a result, the DNS client receives the **NXDomain** error message, informing the user that the queried domain does not exist.

27.3. ADDING A GLOBAL FORWARDER IN THE IDM WEB UI

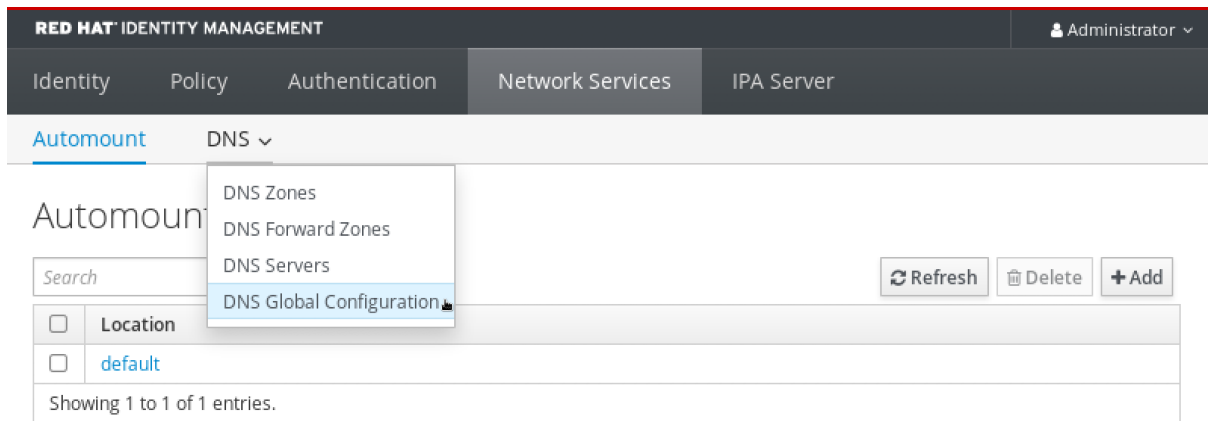
This section describes how to add a global DNS forwarder in the Identity Management (IdM) Web UI.

Prerequisites

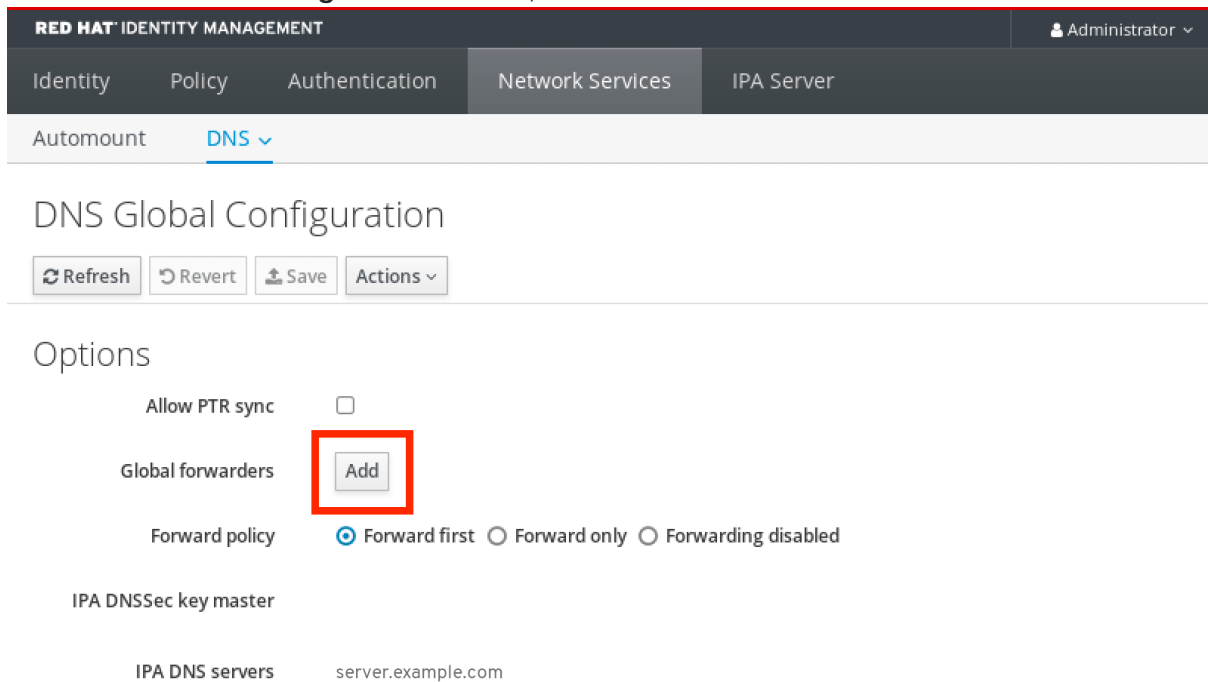
- You are logged in to the IdM WebUI as IdM administrator.
- You know the Internet Protocol (IP) address of the DNS server to forward queries to.

Procedure

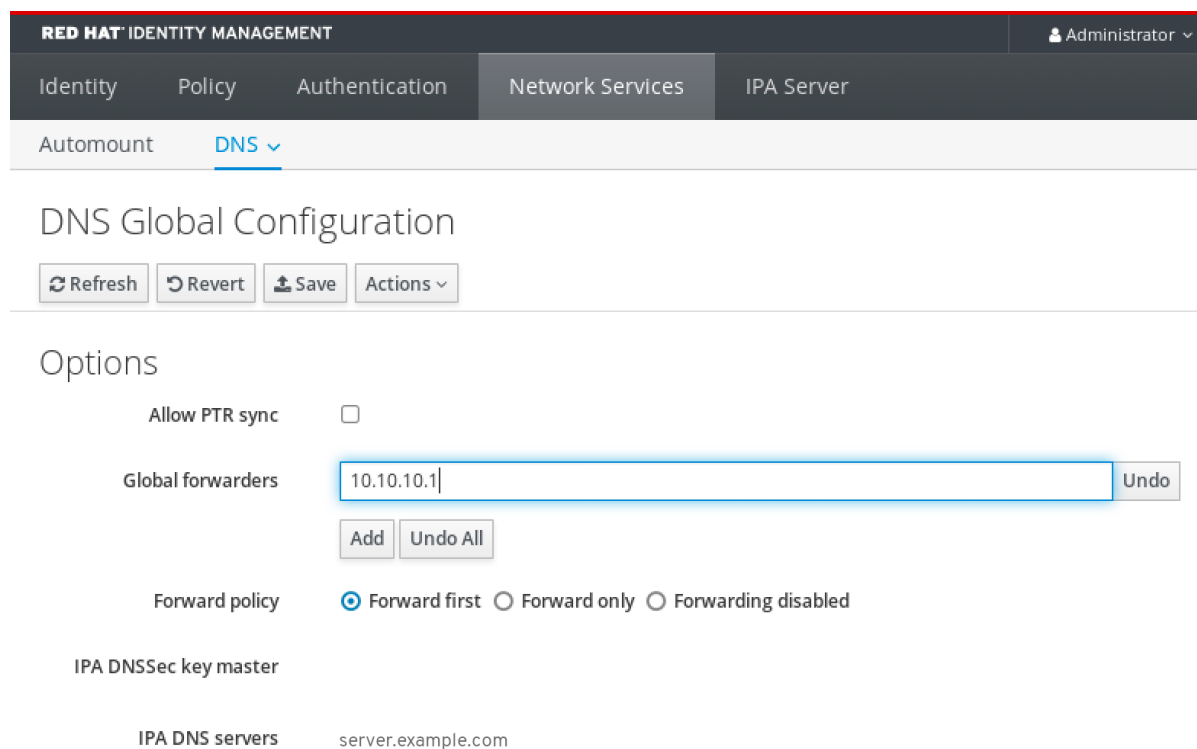
1. In the IdM Web UI, select **Network Services** → **DNS Global Configuration** → **DNS**.



2. In the **DNS Global Configuration** section, click **Add**.



3. Specify the IP address of the DNS server that will receive forwarded DNS queries.



RED HAT IDENTITY MANAGEMENT Administrator ▾

Identity Policy Authentication **Network Services** IPA Server

Automount **DNS ▾**

DNS Global Configuration

[Refresh](#) [Revert](#) [Save](#) [Actions ▾](#)

Options

Allow PTR sync ☐

Global forwarders [Undo](#)

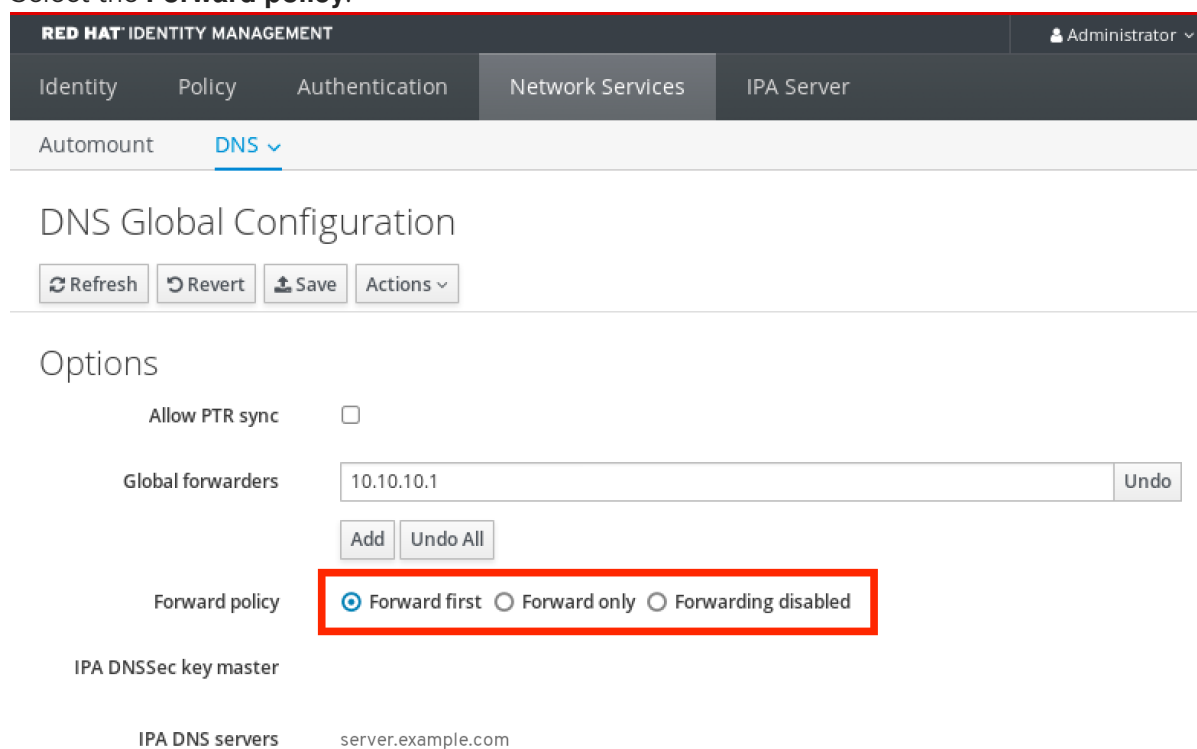
[Add](#) [Undo All](#)

Forward policy ☒ Forward first ☐ Forward only ☐ Forwarding disabled

IPA DNSSec key master

IPA DNS servers server.example.com

4. Select the **Forward policy**.



RED HAT IDENTITY MANAGEMENT Administrator ▾

Identity Policy Authentication **Network Services** IPA Server

Automount **DNS ▾**

DNS Global Configuration

[Refresh](#) [Revert](#) [Save](#) [Actions ▾](#)

Options

Allow PTR sync ☐

Global forwarders [Undo](#)

[Add](#) [Undo All](#)

Forward policy ☒ Forward first ☐ Forward only ☐ Forwarding disabled

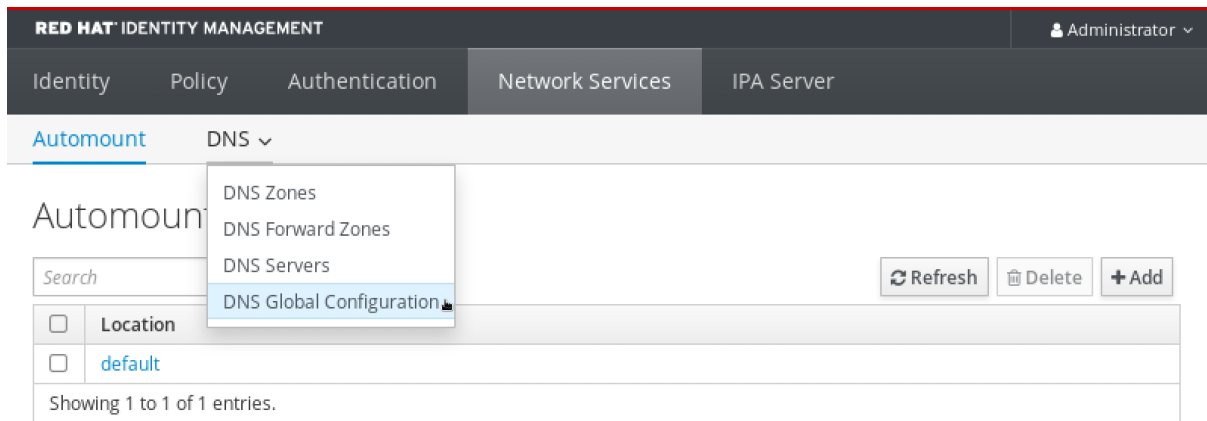
IPA DNSSec key master

IPA DNS servers server.example.com

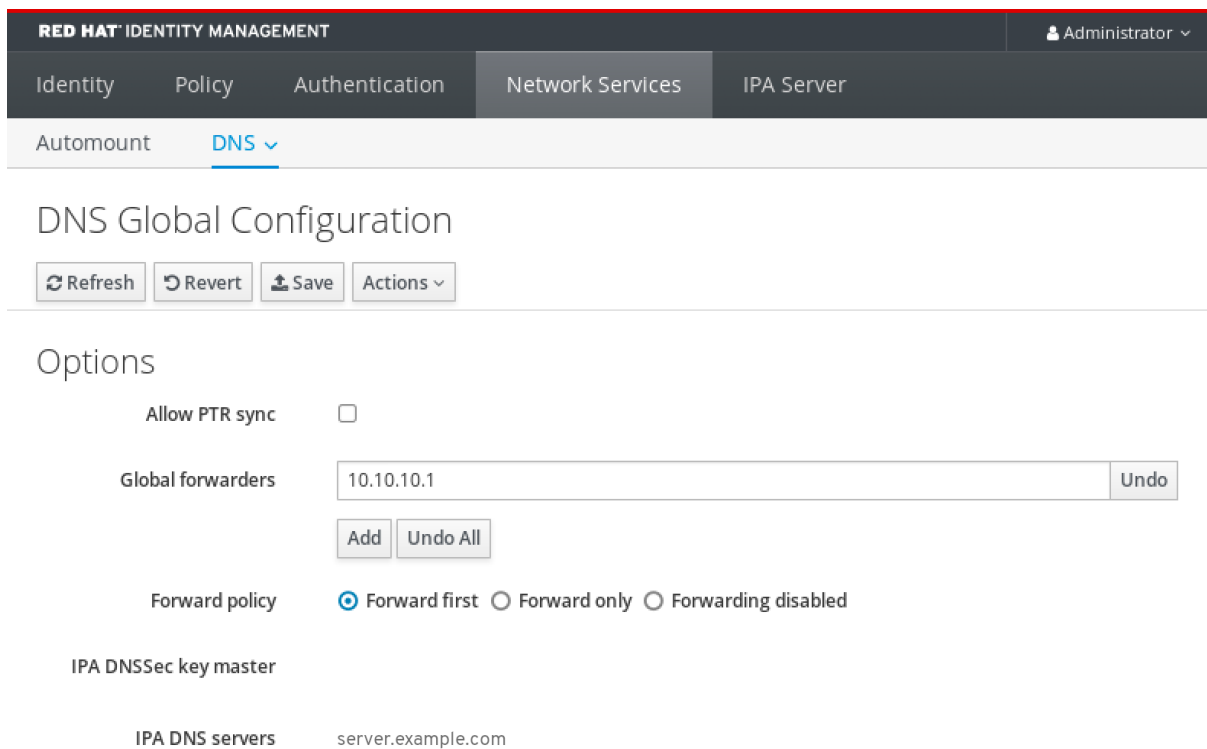
5. Click **Save** at the top of the window.

Verification steps

1. Select **Network Services** → **DNS Global Configuration** → **DNS**.



2. Verify that the global forwarder, with the forward policy you specified, is present and enabled in the IdM Web UI.



27.4. ADDING A GLOBAL FORWARDER IN THE CLI

This section describes how to add a global DNS forwarder from the command line interface (CLI).

Prerequisites

- You are logged in as IdM administrator.
- You know the Internet Protocol (IP) address of the DNS server to forward queries to.

Procedure

- Use the **ipa dnsconfig-mod** command to add a new global forwarder. Specify the IP address of the DNS forwarder with the **--forwarder** option.

```
[user@server ~]$ ipa dnsconfig-mod --forwarder=10.10.0.1
Server will check DNS forwarder(s).
```

This may take some time, please wait ...
 Global forwarders: 10.10.0.1
 IPA DNS servers: server.example.com

Verification steps

- Use the **dnsconfig-show** command to display global forwarders.

```
[user@server ~]$ ipa dnsconfig-show
Global forwarders: 10.10.0.1
IPA DNS servers: server.example.com
```

27.5. ADDING A DNS FORWARD ZONE IN THE IDM WEB UI

This section describes how to add a DNS forward zone in the Identity Management (IdM) Web UI.



IMPORTANT

Do not use forward zones unless absolutely required. Forward zones are not a standard solution, and using them can lead to unexpected and problematic behavior. If you must use forward zones, limit their use to overriding a global forwarding configuration.

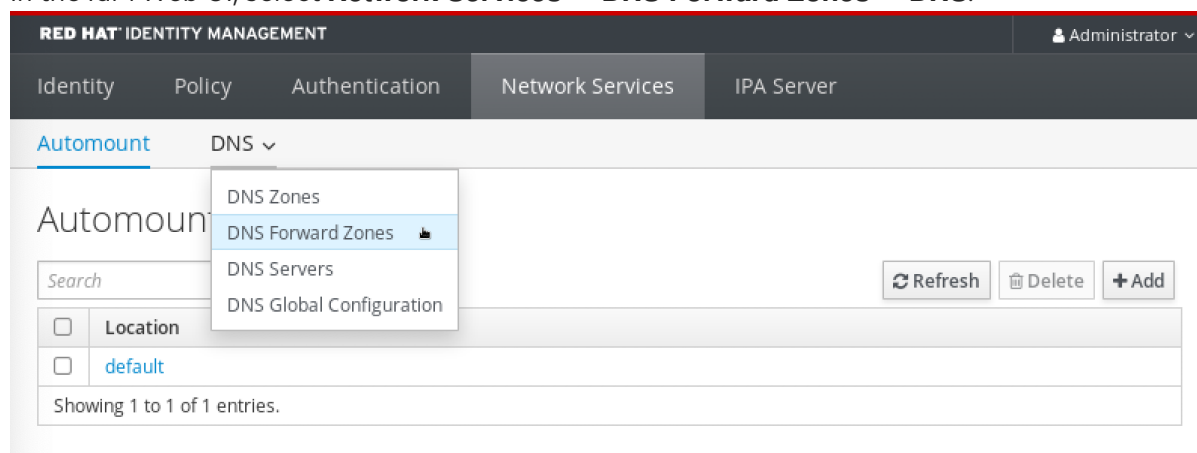
When creating a new DNS zone, Red Hat recommends to always use standard DNS delegation using nameserver (NS) records and to avoid forward zones. In most cases, using a global forwarder is sufficient, and forward zones are not necessary.

Prerequisites

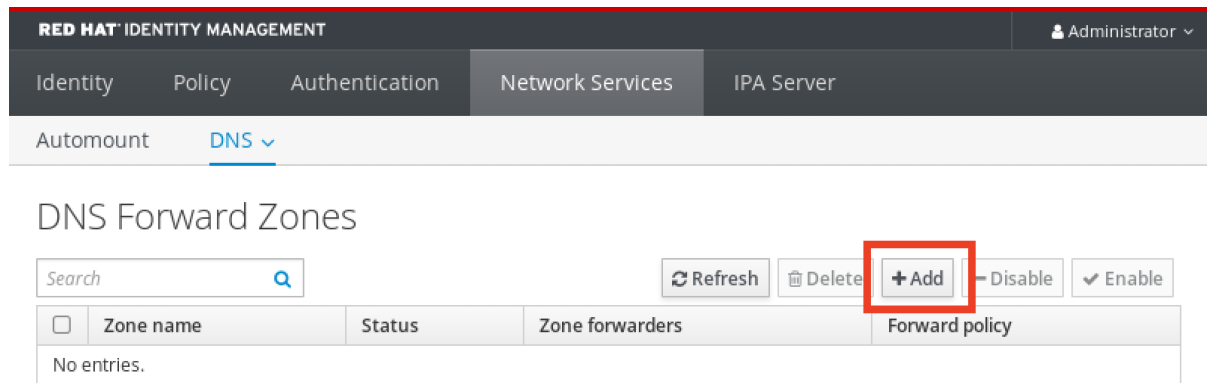
- You are logged in to the IdM WebUI as IdM administrator.
- You know the Internet Protocol (IP) address of the DNS server to forward queries to.

Procedure

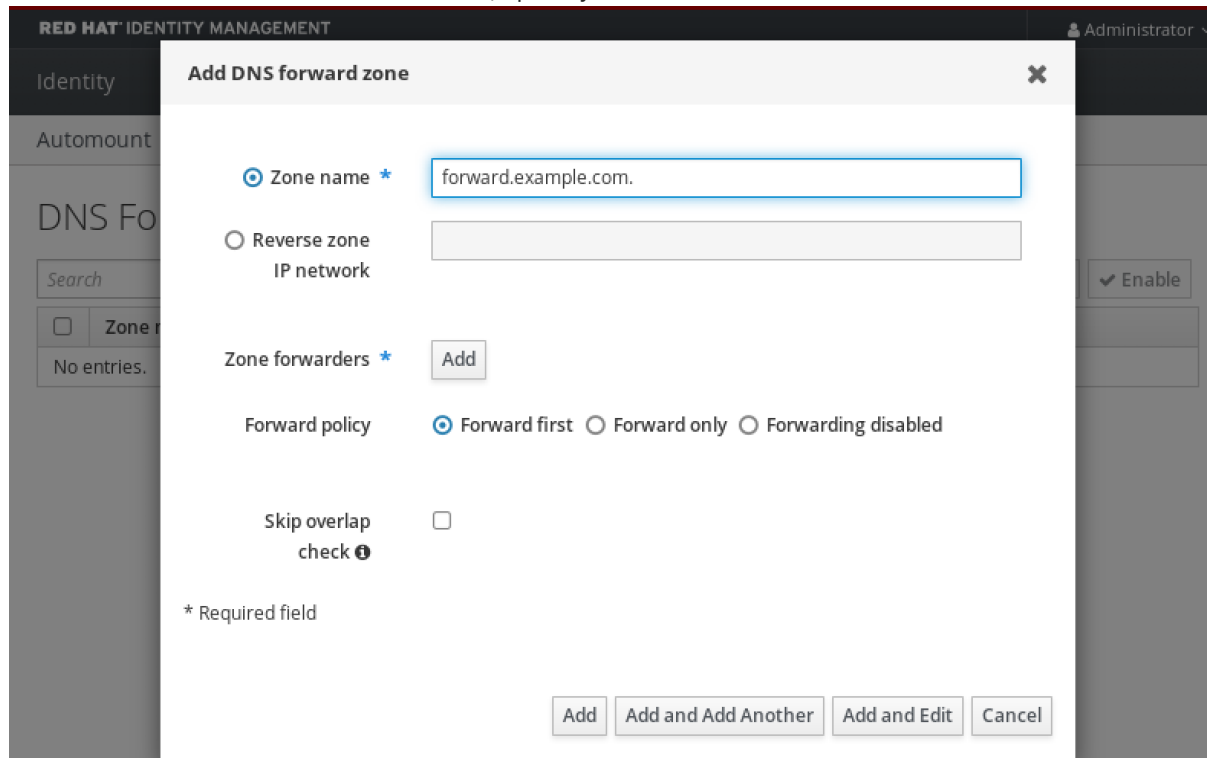
1. In the IdM Web UI, select **Network Services → DNS Forward Zones → DNS**.



2. In the **DNS Forward Zones** section, click **Add**.



3. In the **Add DNS forward zone** window, specify the forward zone name.



4. Click the **Add** button and specify the IP address of a DNS server to receive the forwarding request. You can specify multiple forwarders per forward zone.

Add DNS forward zone

☒ Zone name * forward.example.com.

☐ Reverse zone IP network

Zone forwarders * 10.10.0.14 Undo

Add

Forward policy ☒ Forward first ☐ Forward only ☐ Forwarding disabled

Skip overlap check ☐

* Required field

Add Add and Add Another Add and Edit Cancel

5. Select the **Forward policy**.

Add DNS forward zone

☒ Zone name * forward.example.com

☐ Reverse zone IP network

Zone forwarders * 10.10.0.14 Undo

Add

Forward policy ☒ Forward first ☐ Forward only ☐ Forwarding disabled

Skip overlap check ☐

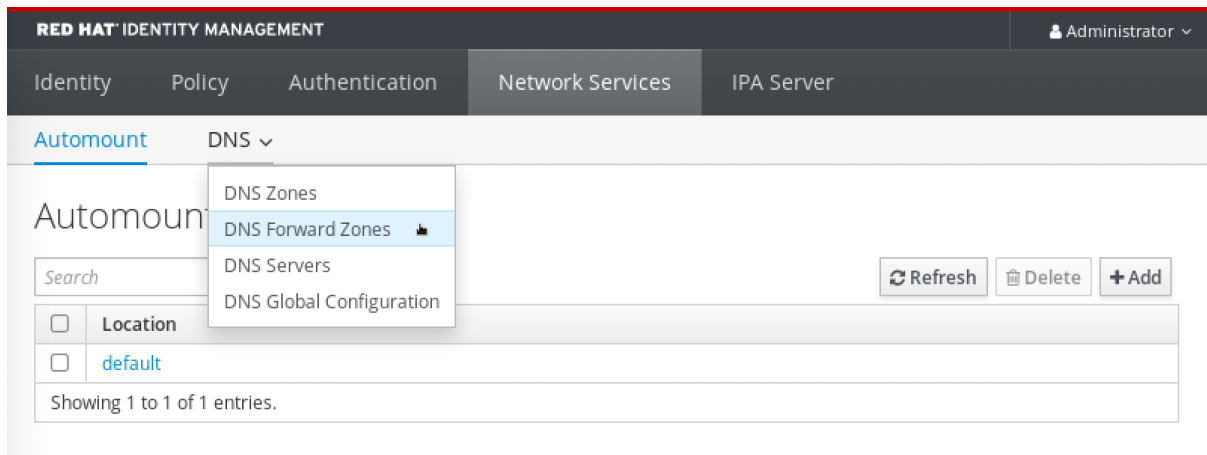
* Required field

Add Add and Add Another Add and Edit Cancel

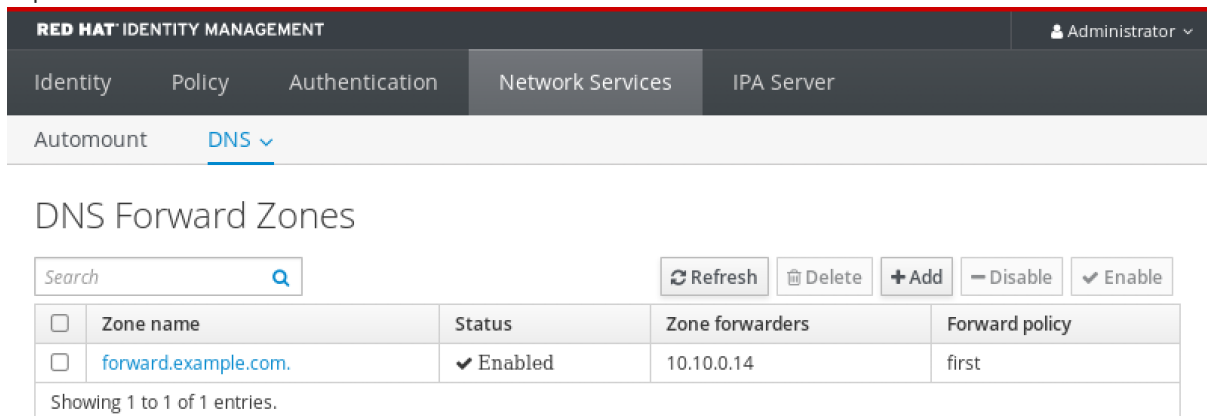
6. Click **Add** at the bottom of the window to add the new forward zone.

Verification steps

1. In the IdM Web UI, select **Network Services** → **DNS Forward Zones** → **DNS**.

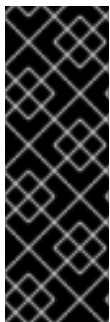


2. Verify that the forward zone you created, with the forwarders and forward policy you specified, is present and enabled in the IdM Web UI.



27.6. ADDING A DNS FORWARD ZONE IN THE CLI

This section describes how to add a DNS forward zone from the command line interface (CLI).



IMPORTANT

Do not use forward zones unless absolutely required. Forward zones are not a standard solution, and using them can lead to unexpected and problematic behavior. If you must use forward zones, limit their use to overriding a global forwarding configuration.

When creating a new DNS zone, Red Hat recommends to always use standard DNS delegation using nameserver (NS) records and to avoid forward zones. In most cases, using a global forwarder is sufficient, and forward zones are not necessary.

Prerequisites

- You are logged in as IdM administrator.
- You know the Internet Protocol (IP) address of the DNS server to forward queries to.

Procedure

- Use the **dnsforwardzone-add** command to add a new forward zone. Specify at least one forwarder with the **--forwarder** option if the forward policy is not **none**, and specify the forward policy with the **--forward-policy** option.

■

```
[user@server ~]$ ipa dnsforwardzone-add forward.example.com. --
forwarder=10.10.0.14 --forwarder=10.10.1.15 --forward-policy=first
```

```
Zone name: forward.example.com.
Zone forwarders: 10.10.0.14, 10.10.1.15
Forward policy: first
```

Verification steps

- Use the **dnsforwardzone-show** command to display the DNS forward zone you just created.

```
[user@server ~]$ ipa dnsforwardzone-show forward.example.com.
```

```
Zone name: forward.example.com.
Zone forwarders: 10.10.0.14, 10.10.1.15
Forward policy: first
```

27.7. ESTABLISHING A DNS GLOBAL FORWARDER IN IDM USING ANSIBLE

This section describes how an Identity Management (IdM) administrator can use an Ansible playbook to establish a DNS Global Forwarder in IdM.

In the example procedure below, the IdM administrator creates a DNS global forwarder to a DNS server with an Internet Protocol (IP) v4 address of **8.8.6.6** and IPv6 address of **2001:4860:4860::8800** on port **53**.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller. This is the host on which you execute the steps in the procedure.
- You know the IdM administrator password.

Procedure

1. Navigate to the **/usr/share/doc/ansible-freeipa/playbooks/dnsconfig** directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/dnsconfig
```

2. Open your inventory file and make sure that the IdM server that you want to configure is listed in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

3. Make a copy of the **set-configuration.yml** Ansible playbook file. For example:

```
$ cp set-configuration.yml establish-global-forwarder.yml
```

4. Open the **establish-global-forwarder.yml** file for editing.

5. Adapt the file by setting the following variables:

- a. Change the **name** variable for the playbook to **Playbook to establish a global forwarder in IdM DNS**.
- b. In the **tasks** section, change the **name** of the task to **Create a DNS global forwarder to 8.8.6.6 and 2001:4860:4860::8800**.
- c. In the **forwarders** section of the **ipadnsconfig** portion:
 - i. Change the first **ip_address** value to the IPv4 address of the global forwarder: **8.8.6.6**.
 - ii. Change the second **ip_address** value to the IPv6 address of the global forwarder: **2001:4860:4860::8800**.
 - iii. Verify the **port** value is set to **53**.
- d. Change the **forward_policy** to **first**.
This the modified Ansible playbook file for the current example:

```
---
- name: Playbook to establish a global forwarder in IdM DNS
  hosts: ipaserver
  become: true

  tasks:
  - name: Create a DNS global forwarder to 8.8.6.6 and 2001:4860:4860::8800
    ipadnsconfig:
      forwarders:
        - ip_address: 8.8.6.6
        - ip_address: 2001:4860:4860::8800
      port: 53
      forward_policy: first
      allow_sync_ptr: yes
```

6. Save the file.

7. Run the playbook:

```
$ ansible-playbook -v -i inventory.file establish-global-forwarder.yml
```

Additional resources

- You can see more sample Ansible playbooks for the **ansible-freeipa ipadnsconfig** module in the **README-dnsconfig.md** Markdown file available in the **/usr/share/doc/ansible-freeipa/** directory. The file also contains the definitions of **ipadnsconfig** variables.

27.8. ENSURING THE PRESENCE OF A DNS GLOBAL FORWARDER IN IDM USING ANSIBLE

This section describes how an Identity Management (IdM) administrator can use an Ansible playbook to ensure the presence of a DNS global forwarder in IdM. In the example procedure below, the IdM administrator ensures the presence of a DNS global forwarder to a DNS server with an Internet Protocol (IP) v4 address of **7.7.9.9** and IP v6 address of **2001:db8::1:0** on port **53**.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller. This is the host on which you execute the steps in the procedure.
- You know the IdM administrator password.

Procedure

1. Navigate to the `/usr/share/doc/ansible-freeipa/playbooks/dnsconfig` directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/dnsconfig
```

2. Open your inventory file and make sure that the IdM server that you want to configure is listed in the `[ipaserver]` section. For example, to instruct Ansible to configure `server.idm.example.com`, enter:

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

3. Make a copy of the `forwarders-absent.yml` Ansible playbook file. For example:

```
$ cp forwarders-absent.yml ensure-presence-of-a-global-forwarder.yml
```

4. Open the `ensure-presence-of-a-global-forwarder.yml` file for editing.

5. Adapt the file by setting the following variables:

- a. Change the `name` variable for the playbook to **Playbook to ensure the presence of a global forwarder in IdM DNS**.
- b. In the `tasks` section, change the `name` of the task to **Ensure the presence of a DNS global forwarder to 7.7.9.9 and 2001:db8::1:0 on port 53**.
- c. In the `forwarders` section of the `ipadnsconfig` portion:
 - i. Change the first `ip_address` value to the IPv4 address of the global forwarder: **7.7.9.9**.
 - ii. Change the second `ip_address` value to the IPv6 address of the global forwarder: **2001:db8::1:0**.
 - iii. Verify the `port` value is set to **53**.
- d. Change the `state` to **present**.
This the modified Ansible playbook file for the current example:

```
---
- name: Playbook to ensure the presence of a global forwarder in IdM DNS
  hosts: ipaserver
  become: true

  tasks:
    - name: Ensure the presence of a DNS global forwarder to 7.7.9.9 and 2001:db8::1:0 on port 53
      ipadnsconfig:
```

```

forwarders:
  - ip_address: 7.7.9.9
  - ip_address: 2001:db8::1:0
    port: 53
    state: present

```

6. Save the file.

7. Run the playbook:

```
$ ansible-playbook -v -i inventory.file ensure-presence-of-a-global-forwarder.yml
```

Additional resources

- You can see more sample Ansible playbooks for the **ansible-freeipa ipadnsconfig** module in the **README-dnsconfig.md** Markdown file available in the `/usr/share/doc/ansible-freeipa/` directory. The file also contains the definitions of **ipadnsconfig** variables.

27.9. ENSURING THE ABSENCE OF A DNS GLOBAL FORWARDER IN IDM USING ANSIBLE

This section describes how an Identity Management (IdM) administrator can use an Ansible playbook to ensure the absence of a DNS global forwarder in IdM. In the example procedure below, the IdM administrator ensures the absence of a DNS global forwarder to a DNS server with an Internet Protocol (IP) v4 address of **8.8.6.6** and IP v6 address of **2001:4860:4860::8800** on port **53**.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller. This is the host on which you execute the steps in the procedure.
- You know the IdM administrator password.

Procedure

- Navigate to the `/usr/share/doc/ansible-freeipa/playbooks/dnsconfig` directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/dnsconfig
```

- Open your inventory file and make sure that the IdM server that you want to configure is listed in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

- Make a copy of the **forwarders-absent.yml** Ansible playbook file. For example:

```
$ cp forwarders-absent.yml ensure-absence-of-a-global-forwarder.yml
```

- Open the **ensure-absence-of-a-global-forwarder.yml** file for editing.
- Adapt the file by setting the following variables:

- a. Change the **name** variable for the playbook to **Playbook to ensure the absence of a global forwarder in IdM DNS**.
- b. In the **tasks** section, change the **name** of the task to **Ensure the absence of a DNS global forwarder to 8.8.6.6 and 2001:4860:4860::8800 on port 53**.
- c. In the **forwarders** section of the **ipadnsconfig** portion:
 - i. Change the first **ip_address** value to the IPv4 address of the global forwarder: **8.8.6.6**.
 - ii. Change the second **ip_address** value to the IPv6 address of the global forwarder: **2001:4860:4860::8800**.
 - iii. Verify the **port** value is set to **53**.
- d. Verify the **state** is set to **absent**.
This is the modified Ansible playbook file for the current example:

```
---
- name: Playbook to ensure the absence of a global forwarder in IdM DNS
  hosts: ipaserver
  become: true

  tasks:
    - name: Ensure the absence of a DNS global forwarder to 8.8.6.6 and
      2001:4860:4860::8800 on port 53
      ipadnsconfig:
        forwarders:
          - ip_address: 8.8.6.6
          - ip_address: 2001:4860:4860::8800
          port: 53
          state: absent
```

6. Save the file.

7. Run the playbook:

```
$ ansible-playbook -v -i inventory.file ensure-absence-of-a-global-forwarder.yml
```

Additional resources

- You can see more sample Ansible playbooks for the **ansible-freeipa ipadnsconfig** module in the **README-dnsconfig.md** Markdown file available in the **/usr/share/doc/ansible-freeipa/** directory. The file also contains the definitions of **ipadnsconfig** variables.

27.10. ENSURING DNS GLOBAL FORWARDERS ARE DISABLED IN IDM USING ANSIBLE

This section describes how an Identity Management (IdM) administrator can use an Ansible playbook to ensure DNS Global Forwarders are disabled in IdM. In the example procedure below, the IdM administrator ensures that the forwarding policy for the global forwarder is set to **none**, which effectively disables the global forwarder.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller. This is the host on which you execute the steps in the procedure.
- You know the IdM administrator password.

Procedure

1. Navigate to the **/usr/share/doc/ansible-freeipa/playbooks/dnsconfig** directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/dnsconfig
```

2. Open your inventory file and make sure that the IdM server that you want to configure is listed in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

3. Verify the contents of the **disable-global-forwarders.yml** Ansible playbook file which is already configured to disable all DNS global forwarders. For example:

```
$ cat disable-global-forwarders.yml
---
- name: Playbook to disable global DNS forwarders
  hosts: ipaserver
  become: true

  tasks:
    - name: Disable global forwarders.
      ipadnsconfig:
        forward_policy: none
```

4. Run the playbook:

```
$ ansible-playbook -v -i inventory.file disable-global-forwarders.yml
```

Additional resources

- You can see more sample Ansible playbooks for the **ansible-freeipa ipadnsconfig** module in the **README-dnsconfig.md** Markdown file available in the **/usr/share/doc/ansible-freeipa/** directory. The file also contains the definitions of **ipadnsconfig** variables.

27.11. ENSURING THE PRESENCE OF A DNS FORWARD ZONE IN IDM USING ANSIBLE

This section describes how an Identity Management (IdM) administrator can use an Ansible playbook to ensure the presence of a DNS Forward Zone in IdM. In the example procedure below, the IdM administrator ensures the presence of a DNS forward zone for **example.com** to a DNS server with an Internet Protocol (IP) address of **8.8.8.8**.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller. This is the host on which you execute the steps in the procedure.
- You know the IdM administrator password.

Procedure

1. Navigate to the **/usr/share/doc/ansible-freeipa/playbooks/dnsconfig** directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/dnsconfig
```

2. Open your inventory file and make sure that the IdM server that you want to configure is listed in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

3. Make a copy of the **forwarders-absent.yml** Ansible playbook file. For example:

```
$ cp forwarders-absent.yml ensure-presence-forwardzone.yml
```

4. Open the **ensure-presence-forwardzone.yml** file for editing.

5. Adapt the file by setting the following variables:

- a. Change the **name** variable for the playbook to **Playbook to ensure the presence of a dnsforwardzone in IdM DNS**.
- b. In the **tasks** section, change the **name** of the task to **Ensure presence of a dnsforwardzone for example.com to 8.8.8.8**.
- c. In the **tasks** section, change the **ipadnsconfig** heading to **ipadnsforwardzone**.
- d. In the **ipadnsforwardzone** section:
 - i. Add the **ipaadmin_password** variable and set it to your IdM administrator password.
 - ii. Add the **name** variable and set it to **example.com**.
 - iii. In the **forwarders** section:
 - A. Remove the **ip_address** and **port** lines.
 - B. Add the IP address of the DNS server to receive forwarded requests by specifying it after a dash:

```
- 8.8.8.8
```

- iv. Add the **forwardpolicy** variable and set it to **first**.
- v. Add the **skip_overlap_check** variable and set it to **true**.
- vi. Change the **state** variable to **present**.

This the modified Ansible playbook file for the current example:

```

---
- name: Playbook to ensure the presence of a dnsforwardzone in IdM DNS
  hosts: ipaserver
  become: true

  tasks:
  - name: Ensure the presence of a dnsforwardzone for example.com to 8.8.8.8
    ipadnsforwardzone:
      ipaadmin_password: password01
      name: example.com
      forwarders:
        - 8.8.8.8
      forwardpolicy: first
      skip_overlap_check: true
      state: present

```

6. Save the file.

7. Run the playbook:

```
$ ansible-playbook -v -i inventory.file ensure-presence-forwardzone.yml
```

Additional resources

- You can see more sample Ansible playbooks for the **ansible-freeipa ipadnsforwardzone** module in the **README-dnsforwardzone.md** Markdown file available in the **/usr/share/doc/ansible-freeipa/** directory. The file also contains the definitions of **ipadnsforwardzone** variables.

27.12. ENSURING A DNS FORWARD ZONE HAS MULTIPLE FORWARDERS IN IDM USING ANSIBLE

This section describes how an Identity Management (IdM) administrator can use an Ansible playbook to ensure a DNS Forward Zone in IdM has multiple forwarders. In the example procedure below, the IdM administrator ensures the DNS forward zone for **example.com** is forwarding to **8.8.8.8** and **4.4.4.4**.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller. This is the host on which you execute the steps in the procedure.
- You know the IdM administrator password.

Procedure

- Navigate to the **/usr/share/doc/ansible-freeipa/playbooks/dnsconfig** directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/dnsconfig
```

- Open your inventory file and make sure that the IdM server that you want to configure is listed in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

3. Make a copy of the **forwarders-absent.yml** Ansible playbook file. For example:

```
$ cp forwarders-absent.yml ensure-presence-multiple-forwarders.yml
```

4. Open the **ensure-presence-multiple-forwarders.yml** file for editing.
5. Adapt the file by setting the following variables:
 - a. Change the **name** variable for the playbook to **Playbook to ensure the presence of multiple forwarders in a dnsforwardzone in IdM DNS**.
 - b. In the **tasks** section, change the **name** of the task to **Ensure presence of 8.8.8.8 and 4.4.4.4 forwarders in dnsforwardzone for example.com**.
 - c. In the **tasks** section, change the **ipadnsconfig** heading to **ipadnsforwardzone**.
 - d. In the **ipadnsforwardzone** section:
 - i. Add the **ipaadmin_password** variable and set it to your IdM administrator password.
 - ii. Add the **name** variable and set it to **example.com**.
 - iii. In the **forwarders** section:
 - A. Remove the **ip_address** and **port** lines.
 - B. Add the IP address of the DNS servers you want to ensure are present, preceded by a dash:


```
- 8.8.8.8
- 4.4.4.4
```
 - iv. Change the state variable to present.

This the modified Ansible playbook file for the current example:

```
---
- name: name: Playbook to ensure the presence of multiple forwarders in a dnsforwardzone
  in IdM DNS
  hosts: ipaserver
  become: true

  tasks:
  - name: Ensure presence of 8.8.8.8 and 4.4.4.4 forwarders in dnsforwardzone for
    example.com
    ipadnsforwardzone:
      ipaadmin_password: password01
      name: example.com
      forwarders:
        - 8.8.8.8
        - 4.4.4.4
      state: present
```


6. Save the file.
7. Run the playbook:

```
$ ansible-playbook -v -i inventory.file ensure-presence-multiple-forwarders.yml
```

Additional resources

- You can see more sample Ansible playbooks for the **ansible-freeipa ipadsforwardzone** module in the **README-dnsforwardzone.md** Markdown file available in the **/usr/share/doc/ansible-freeipa/** directory. The file also contains the definitions of **ipadsforwardzone** variables.

27.13. ENSURING A DNS FORWARD ZONE IS DISABLED IN IDM USING ANSIBLE

This section describes how an Identity Management (IdM) administrator can use an Ansible playbook to ensure a DNS Forward Zone is disabled in IdM. In the example procedure below, the IdM administrator ensures the DNS forward zone for **example.com** is disabled.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller. This is the host on which you execute the steps in the procedure.
- You know the IdM administrator password.

Procedure

1. Navigate to the **/usr/share/doc/ansible-freeipa/playbooks/dnsconfig** directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/dnsconfig
```

2. Open your inventory file and make sure that the IdM server that you want to configure is listed in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

3. Make a copy of the **forwarders-absent.yml** Ansible playbook file. For example:

```
$ cp forwarders-absent.yml ensure-disabled-forwardzone.yml
```

4. Open the **ensure-disabled-forwardzone.yml** file for editing.
5. Adapt the file by setting the following variables:
 - a. Change the **name** variable for the playbook to **Playbook to ensure a dnsforwardzone is disabled in IdM DNS**.
 - b. In the **tasks** section, change the **name** of the task to **Ensure a dnsforwardzone for example.com is disabled**.

- c. In the **tasks** section, change the **ipadnsconfig** heading to **ipadnsforwardzone**.
- d. In the **ipadnsforwardzone** section:
 - i. Add the **ipaadmin_password** variable and set it to your IdM administrator password.
 - ii. Add the **name** variable and set it to **example.com**.
 - iii. Remove the entire **forwarders** section.
 - iv. Change the **state** variable to **disabled**.

This the modified Ansible playbook file for the current example:

```
---
- name: Playbook to ensure a dnsforwardzone is disabled in IdM DNS
  hosts: ipaserver
  become: true

  tasks:
  - name: Ensure a dnsforwardzone for example.com is disabled
    ipadnsforwardzone:
      ipaadmin_password: password01
      name: example.com
      state: disabled
```

6. Save the file.
7. Run the playbook:

```
$ ansible-playbook -v -i inventory.file ensure-disabled-forwardzone.yml
```

Additional resources

- You can see more sample Ansible playbooks for the **ansible-freeipa ipadnsforwardzone** module in the **README-dnsforwardzone.md** Markdown file available in the **/usr/share/doc/ansible-freeipa/** directory. The file also contains the definitions of **ipadnsforwardzone** variables.

27.14. ENSURING THE ABSENCE OF A DNS FORWARD ZONE IN IDM USING ANSIBLE

This section describes how an Identity Management (IdM) administrator can use an Ansible playbook to ensure the absence of a DNS Forward Zone in IdM. In the example procedure below, the IdM administrator ensures the absence of a DNS forward zone for **example.com**.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller. This is the host on which you execute the steps in the procedure.
- You know the IdM administrator password.

Procedure

1. Navigate to the `/usr/share/doc/ansible-freeipa/playbooks/dnsconfig` directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/dnsconfig
```

2. Open your inventory file and make sure that the IdM server that you want to configure is listed in the `[ipaserver]` section. For example, to instruct Ansible to configure `server.idm.example.com`, enter:

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

3. Make a copy of the `forwarders-absent.yml` Ansible playbook file. For example:

```
$ cp forwarders-absent.yml ensure-absence-forwardzone.yml
```

4. Open the `ensure-absence-forwardzone.yml` file for editing.

5. Adapt the file by setting the following variables:

- a. Change the `name` variable for the playbook to **Playbook to ensure the absence of a dnsforwardzone in IdM DNS**.
- b. In the `tasks` section, change the `name` of the task to **Ensure the absence of a dnsforwardzone for example.com**.
- c. In the `tasks` section, change the `ipadnsconfig` heading to `ipadnsforwardzone`.
- d. In the `ipadnsforwardzone` section:
 - i. Add the `ipaadmin_password` variable and set it to your IdM administrator password.
 - ii. Add the `name` variable and set it to `example.com`.
 - iii. Remove the entire `forwarders` section.
 - iv. Leave the `state` variable as `absent`.

This the modified Ansible playbook file for the current example:

```
---
- name: Playbook to ensure the absence of a dnsforwardzone in IdM DNS
  hosts: ipaserver
  become: true

  tasks:
    - name: Ensure the absence of a dnsforwardzone for example.com
      ipadnsforwardzone:
        ipaadmin_password: password01
        name: example.com
        state: absent
```

6. Save the file.
7. Run the playbook:

```
$ ansible-playbook -v -i inventory.file ensure-absence-forwardzone.yml
```

Additional resources

- You can see more sample Ansible playbooks for the **ansible-freeipa ipadnsforwardzone** module in the **README-dnsforwardzone.md** Markdown file available in the **/usr/share/doc/ansible-freeipa/** directory. The file also contains the definitions of **ipadnsforwardzone** variables.

CHAPTER 28. USING ANSIBLE TO MANAGE DNS RECORDS IN IDM

This chapter describes how to manage DNS records in Identity Management (IdM) using an Ansible playbook. As an IdM administrator, you can add, modify, and delete DNS records in IdM. The chapter contains the following sections:

- [Ensuring the presence of A and AAAA DNS records in IdM using Ansible](#)
- [Ensuring the presence of A and PTR DNS records in IdM using Ansible](#)
- [Ensuring the presence of multiple DNS records in IdM using Ansible](#)
- [Ensuring the presence of multiple CNAME records in IdM using Ansible](#)
- [Ensuring the presence of an SRV record in IdM using Ansible](#)

28.1. DNS RECORDS IN IDM

Identity Management (IdM) supports many different DNS record types. The following four are used most frequently:

A

This is a basic map for a host name and an IPv4 address. The record name of an A record is a host name, such as **www**. The **IP Address** value of an A record is an IPv4 address, such as **192.0.2.1**. For more information about A records, see [RFC 1035](#).

AAAA

This is a basic map for a host name and an IPv6 address. The record name of an AAAA record is a host name, such as **www**. The **IP Address** value is an IPv6 address, such as **2001:DB8::1111**. For more information about AAAA records, see [RFC 3596](#).

SRV

Service (SRV) resource records map service names to the DNS name of the server that is providing that particular service. For example, this record type can map a service like an LDAP directory to the server which manages it.

The record name of an SRV record has the format **_service._protocol**, such as **_ldap._tcp**. The configuration options for SRV records include priority, weight, port number, and host name for the target service.

For more information about SRV records, see [RFC 2782](#).

PTR

A pointer record (PTR) adds a reverse DNS record, which maps an IP address to a domain name.



NOTE

All reverse DNS lookups for IPv4 addresses use reverse entries that are defined in the **in-addr.arpa** domain. The reverse address, in human-readable form, is the exact reverse of the regular IP address, with the **in-addr.arpa** domain appended to it. For example, for the network address **192.0.2.0/24**, the reverse zone is **2.0.192.in-addr.arpa**.

The record name of a PTR must be in the standard format specified in [RFC 1035](#), extended in [RFC 2317](#), and [RFC 3596](#). The host name value must be a canonical host name of the host for which you want to create the record.



NOTE

Reverse zones can also be configured for IPv6 addresses, with zones in the **.ip6.arpa.** domain. For more information about IPv6 reverse zones, see [RFC 3596](#).

When adding DNS resource records, note that many of the records require different data. For example, a CNAME record requires a host name, while an A record requires an IP address. In the IdM Web UI, the fields in the form for adding a new record are updated automatically to reflect what data is required for the currently selected type of record.

28.2. COMMON IPA DNSRECORD-* OPTIONS

This section describes the options you can use when adding, modifying and deleting the most common DNS resource record types to Identity Management (IdM):

- A (IPv4)
- AAAA (IPv6)
- SRV
- PTR

In **Bash**, you can define multiple entries by listing the values in a comma-separated list inside curly braces, such as **--option={val1,val2,val3}**.

Table 28.1. General Record Options

Option	Description
--ttl=number	Sets the time to live for the record.
--structured	Parses the raw DNS records and returns them in a structured format.

Table 28.2. "A" record options

Option	Description	Examples
--a-rec=ARECORD	Passes a single A record or a list of A records.	ipa dnsrecord-add idm.example.com host1 --a-rec=192.168.122.123
	Can create a wildcard A record with a given IP address.	ipa dnsrecord-add idm.example.com "*" --a-rec=192.168.122.123 ^[a]

Option	Description	Examples
--a-ip-address=string	Gives the IP address for the record. When creating a record, the option to specify the A record value is --a-rec . However, when modifying an A record, the --a-rec option is used to specify the current value for the A record. The new value is set with the --a-ip-address option.	ipa dnsrecord-mod idm.example.com --a-rec 192.168.122.123 --a-ip- address 192.168.122.124
[a] The example creates a wildcard A record with the IP address of 192.0.2.123.		

Table 28.3. "AAAA" record options

Option	Description	Example
--aaaa-rec=AAAARECORD	Passes a single AAAA (IPv6) record or a list of AAAA records.	ipa dnsrecord-add idm.example.com www -- aaaa-rec 2001:db8::1231:5675
--aaaa-ip-address=string	Gives the IPv6 address for the record. When creating a record, the option to specify the A record value is --aaaa-rec . However, when modifying an A record, the --aaaa-rec option is used to specify the current value for the A record. The new value is set with the --a-ip-address option.	ipa dnsrecord-mod idm.example.com --aaaa-rec 2001:db8::1231:5675 --aaaa- ip-address 2001:db8::1231:5676

Table 28.4. "PTR" record options

Option	Description	Example
--ptr-rec=PTRRECORD	Passes a single PTR record or a list of PTR records. When adding the reverse DNS record, the zone name used with the ipa dnsrecord-add command is reversed, compared to the usage for adding other DNS records. Typically, the host IP address is the last octet of the IP address in a given network. The first example on the right adds a PTR record for server4.idm.example.com with IPv4 address 192.168.122.4 . The second example adds a reverse DNS entry to the 0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa . IPv6 reverse zone for the host server2.example.com with the IP address 2001:DB8::1111 .	ipa dnsrecord-add 122.168.192.in-addr.arpa 4 -- ptr-rec server4.idm.example.com. \$ ipa dnsrecord-add 0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.i p6.arpa. 1.1.1.0.0.0.0.0.0.0.0.0.0.0 -- ptr-rec server2.idm.example.com.
--ptr-hostname=string	Gives the host name for the record.	

Table 28.5. "SRV" Record Options

Option	Description	Example
--srv-rec=SRVRECORD	Passes a single SRV record or a list of SRV records. In the examples on the right, <code>_ldap._tcp</code> defines the service type and the connection protocol for the SRV record. The --srv-rec option defines the priority, weight, port, and target values. The weight values of 51 and 49 in the examples add up to 100 and represent the probability, in percentages, that a particular record is used.	<pre># ipa dnsrecord-add idm.example.com _ldap._tcp --srv-rec="0 51 389 server1.idm.example.com."</pre> <pre># ipa dnsrecord-add server.idm.example.com _ldap._tcp --srv-rec="1 49 389 server2.idm.example.com."</pre>
--srv-priority=number	Sets the priority of the record. There can be multiple SRV records for a service type. The priority (0 - 65535) sets the rank of the record; the lower the number, the higher the priority. A service has to use the record with the highest priority first.	<pre># ipa dnsrecord-mod server.idm.example.com _ldap._tcp --srv-rec="1 49 389 server2.idm.example.com." --srv-priority=0</pre>
--srv-weight=number	Sets the weight of the record. This helps determine the order of SRV records with the same priority. The set weights should add up to 100, representing the probability (in percentages) that a particular record is used.	<pre># ipa dnsrecord-mod server.idm.example.com _ldap._tcp --srv-rec="0 49 389 server2.idm.example.com." --srv-weight=60</pre>
--srv-port=number	Gives the port for the service on the target host.	<pre># ipa dnsrecord-mod server.idm.example.com _ldap._tcp --srv-rec="0 60 389 server2.idm.example.com." --srv-port=636</pre>
--srv-target=string	Gives the domain name of the target host. This can be a single period (.) if the service is not available in the domain.	

Additional resources

- For more information on how to use **ipa dnsrecord-add** and which DNS record types are supported by IdM, run the **ipa dnsrecord-add --help** command.

28.3. ENSURING THE PRESENCE OF A AND AAAA DNS RECORDS IN IDM USING ANSIBLE

This section shows how an Identity Management (IdM) administrator can use an Ansible playbook to ensure that A and AAAA records for a particular IdM host are present. In the example used in the procedure below, an IdM administrator ensures the presence of A and AAAA records for **host1** in the **idm.example.com** DNS zone.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller. This is the host on which you execute the steps in the procedure.
- You know the IdM administrator password.
- The **idm.example.com** zone exists and is managed by IdM DNS. For more information about adding a primary DNS zone in IdM DNS, see [Using Ansible playbooks to manage IdM DNS zones](#) .

Procedure

1. Navigate to the **/usr/share/doc/ansible-freeipa/playbooks/dnsrecord** directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/dnsrecord
```

2. Open your inventory file and ensure that the IdM server that you want to configure is listed in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

3. Make a copy of the **ensure-A-and-AAAA-records-are-present.yml** Ansible playbook file. For example:

```
$ cp ensure-A-and-AAAA-records-are-present.yml ensure-A-and-AAAA-records-are-present-copy.yml
```

4. Open the **ensure-A-and-AAAA-records-are-present-copy.yml** file for editing.
5. Adapt the file by setting the following variables in the **ipadnsrecord** task section:
 - Set the **ipaadmin_password** variable to your IdM administrator password.
 - Set the **zone_name** variable to **idm.example.com**.
 - In the **records** variable, set the **name** variable to **host1**, and the **a_ip_address** variable to **192.168.122.123**.
 - In the **records** variable, set the **name** variable to **host1**, and the **aaaa_ip_address** variable to **::1**.

This is the modified Ansible playbook file for the current example:

```
---
- name: Ensure A and AAAA records are present
  hosts: ipaserver
  become: true
  gather_facts: false

  tasks:
    # Ensure A and AAAA records are present
    - name: Ensure that 'host1' has A and AAAA records.
      ipadnsrecord:
        ipaadmin_password: Secret123
```

```

zone_name: idm.example.com
records:
- name: host1
  a_ip_address: 192.168.122.123
- name: host1
  aaaa_ip_address: ::1

```

6. Save the file.

7. Run the playbook:

```
$ ansible-playbook -v -i inventory.file ensure-A-and-AAAA-records-are-present-copy.yml
```

Additional resources

- For more information on A and AAAA records, see [DNS records in IdM](#).
- You can see more sample Ansible playbooks for the **ansible-freeipa ipadnsrecord** module in the **README-dnsrecord.md** Markdown file available in the `/usr/share/doc/ansible-freeipa/` directory. The file also contains the definitions of the **ipadnsrecord** variables.
- You can see sample Ansible playbooks for the **ipadnsrecord** module in the `/usr/share/doc/ansible-freeipa/playbooks/dnsrecord` directory.

28.4. ENSURING THE PRESENCE OF A AND PTR DNS RECORDS IN IDM USING ANSIBLE

This section shows how an Identity Management (IdM) administrator can use an Ansible playbook to ensure that an A record for a particular IdM host is present, with a corresponding PTR record. In the example used in the procedure below, an IdM administrator ensures the presence of A and PTR records for **host1** with an IP address of **192.168.122.45** in the **idm.example.com** zone.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller. This is the host on which you execute the steps in the procedure.
- You know the IdM administrator password.
- The **idm.example.com** DNS zone exists and is managed by IdM DNS. For more information about adding a primary DNS zone in IdM DNS, see [Using Ansible playbooks to manage IdM DNS zones](#).

Procedure

1. Navigate to the `/usr/share/doc/ansible-freeipa/playbooks/dnsrecord` directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/dnsrecord
```

2. Open your inventory file and ensure that the IdM server that you want to configure is listed in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

3. Make a copy of the **ensure-dnsrecord-with-reverse-is-present.yml** Ansible playbook file. For example:

```
$ cp ensure-dnsrecord-with-reverse-is-present.yml ensure-dnsrecord-with-reverse-is-present-copy.yml
```

4. Open the **ensure-dnsrecord-with-reverse-is-present-copy.yml** file for editing.
5. Adapt the file by setting the following variables in the **ipadnsrecord** task section:

- Set the **ipaadmin_password** variable to your IdM administrator password.
- Set the **name** variable to **host1**.
- Set the **zone_name** variable to **idm.example.com**.
- Set the **ip_address** variable to **192.168.122.45**.
- Set the **create_reverse** variable to **yes**.

This is the modified Ansible playbook file for the current example:

```
---
- name: Ensure DNS Record is present.
  hosts: ipaserver
  become: true
  gather_facts: false

  tasks:
    # Ensure that dns record is present
    - ipadnsrecord:
        ipaadmin_password: Secret123
        name: host1
        zone_name: idm.example.com
        ip_address: 192.168.122.45
        create_reverse: yes
        state: present
```

6. Save the file.
7. Run the playbook:

```
$ ansible-playbook -v -i inventory.file ensure-dnsrecord-with-reverse-is-present-copy.yml
```

Additional resources

- For more information on A and PTR DNS records, see [DNS records in IdM](#).
- You can see more sample Ansible playbooks for the **ansible-freeipa ipadnsrecord** module in the **README-dnsrecord.md** Markdown file available in the **/usr/share/doc/ansible-freeipa/** directory. The file also contains the definitions of the **ipadnsrecord** variables.

- You can see sample Ansible playbooks for the **ipadnsrecord** module in the **/usr/share/doc/ansible-freeipa/playbooks/dnsrecord** directory.

28.5. ENSURING THE PRESENCE OF MULTIPLE DNS RECORDS IN IDM USING ANSIBLE

This section shows how an Identity Management (IdM) administrator can use an Ansible playbook to ensure that multiple values are associated with a particular IdM DNS record. In the example used in the procedure below, an IdM administrator ensures the presence of multiple A records for **host1** in the **idm.example.com** DNS zone.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller. This is the host on which you execute the steps in the procedure.
- You know the IdM administrator password.
- The **idm.example.com** zone exists and is managed by IdM DNS. For more information about adding a primary DNS zone in IdM DNS, see [Using Ansible playbooks to manage IdM DNS zones](#) .

Procedure

1. Navigate to the **/usr/share/doc/ansible-freeipa/playbooks/dnsrecord** directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/dnsrecord
```

2. Open your inventory file and ensure that the IdM server that you want to configure is listed in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

3. Make a copy of the **ensure-presence-multiple-records.yml** Ansible playbook file. For example:

```
$ cp ensure-presence-multiple-records.yml ensure-presence-multiple-records-copy.yml
```

4. Open the **ensure-presence-multiple-records-copy.yml** file for editing.
5. Adapt the file by setting the following variables in the **ipadnsrecord** task section:

- Set the **ipaadmin_password** variable to your IdM administrator password.
- In the **records** section, set the **name** variable to **host1**.
- In the **records** section, set the **zone_name** variable to **idm.example.com**.
- In the **records** section, set the **a_rec** variable to **192.168.122.112** and to **192.168.122.122**.
- Define a second record in the **records** section:
 - Set the **name** variable to **host1**.

- Set the **zone_name** variable to **idm.example.com**.
- Set the **aaaa_rec** variable to **::1**.

This is the modified Ansible playbook file for the current example:

```
---
- name: Test multiple DNS Records are present.
  hosts: ipaserver
  become: true
  gather_facts: false

  tasks:
    # Ensure that multiple dns records are present
    - ipadnsrecord:
        ipadmin_password: Secret123
        records:
          - name: host1
            zone_name: idm.example.com
            a_rec: 192.168.122.112
            a_rec: 192.168.122.122
          - name: host1
            zone_name: idm.example.com
            aaaa_rec: ::1
```

6. Save the file.
7. Run the playbook:

```
$ ansible-playbook -v -i inventory.file ensure-presence-multiple-records-copy.yml
```

Additional resources

- For more information on A records in DNS, see [DNS records in IdM](#).
- You can see more sample Ansible playbooks for the **ansible-freeipa ipadnsrecord** module in the **README-dnsrecord.md** Markdown file available in the **/usr/share/doc/ansible-freeipa/** directory. The file also contains the definitions of the **ipadnsrecord** variables.
- You can see sample Ansible playbooks for the **ipadnsrecord** module in the **/usr/share/doc/ansible-freeipa/playbooks/dnsrecord** directory.

28.6. ENSURING THE PRESENCE OF MULTIPLE CNAME RECORDS IN IDM USING ANSIBLE

A Canonical Name record (CNAME record) is a type of resource record in the Domain Name System (DNS) that maps one domain name, an alias, to another name, the canonical name.

You may find CNAME records useful when running multiple services from a single IP address: for example, an FTP service and a web service, each running on a different port.

This section shows how an Identity Management (IdM) administrator can use an Ansible playbook to ensure that multiple CNAME records are present in IdM DNS. In the example used in the procedure below, **host03** is both an HTTP server and an FTP server. The IdM administrator ensures the presence

of the **www** and **ftp** CNAME records for the **host03** A record in the **idm.example.com** zone.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller. This is the host on which you execute the steps in the procedure.
- You know the IdM administrator password.
- The **idm.example.com** zone exists and is managed by IdM DNS. For more information about adding a primary DNS zone in IdM DNS, see [Using Ansible playbooks to manage IdM DNS zones](#) .
- The **host03** A record exists in the **idm.example.com** zone.

Procedure

1. Navigate to the **/usr/share/doc/ansible-freeipa/playbooks/dnsrecord** directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/dnsrecord
```

2. Open your inventory file and ensure that the IdM server that you want to configure is listed in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

3. Make a copy of the **ensure-CNAME-record-is-present.yml** Ansible playbook file. For example:

```
$ cp ensure-CNAME-record-is-present.yml ensure-CNAME-record-is-present-copy.yml
```

4. Open the **ensure-CNAME-record-is-present-copy.yml** file for editing.
5. Adapt the file by setting the following variables in the **ipadnsrecord** task section:

- (Optional) Adapt the description provided by the **name** of the play.
- Set the **ipaadmin_password** variable to your IdM administrator password.
- Set the **zone_name** variable to **idm.example.com**.
- In the **records** variable section, set the following variables and values:
 - Set the **name** variable to **www**.
 - Set the **cname_hostname** variable to **host03**.
 - Set the **name** variable to **ftp**.
 - Set the **cname_hostname** variable to **host03**.

This is the modified Ansible playbook file for the current example:

```
---
- name: Ensure that 'www.idm.example.com' and 'ftp.idm.example.com' CNAME records
```

```

point to 'host03.idm.example.com'.
hosts: ipaserver
become: true
gather_facts: false

tasks:
- ipadsnsrecord:
    ipaadmin_password: Secret123
    zone_name: idm.example.com
    records:
    - name: www
      cname_hostname: host03
    - name: ftp
      cname_hostname: host03

```

6. Save the file.

7. Run the playbook:

```
$ ansible-playbook -v -i inventory.file ensure-CNAME-record-is-present.yml
```

Additional resources

- You can see more sample Ansible playbooks for the **ansible-freeipa ipadsnsrecord** module in the **README-dnsrecord.md** Markdown file available in the `/usr/share/doc/ansible-freeipa/` directory. The file also contains the definitions of the **ipadsnsrecord** variables.
- You can see sample Ansible playbooks for the **ipadsnsrecord** module in the `/usr/share/doc/ansible-freeipa/playbooks/dnsrecord` directory.

28.7. ENSURING THE PRESENCE OF AN SRV RECORD IN IDM USING ANSIBLE

A DNS service (SRV) record defines the hostname, port number, transport protocol, priority and weight of a service available in a domain. In Identity Management (IdM), you can use SRV records to locate IdM servers and replicas.

This section shows how an Identity Management (IdM) administrator can use an Ansible playbook to ensure that an SRV record is present in IdM DNS. In the example used in the procedure below, an IdM administrator ensures the presence of the `_kerberos._udp.idm.example.com` SRV record with the value of `10 50 88 idm.example.com`. This sets the following values:

- It sets the priority of the service to 10.
- It sets the weight of the service to 50.
- It sets the port to be used by the service to 88.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible controller. This is the host on which you execute the steps in the procedure.
- You know the IdM administrator password.

- The **idm.example.com** zone exists and is managed by IdM DNS. For more information about adding a primary DNS zone in IdM DNS, see [Using Ansible playbooks to manage IdM DNS zones](#) .

Procedure

1. Navigate to the **/usr/share/doc/ansible-freeipa/playbooks/dnsrecord** directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/dnsrecord
```

2. Open your inventory file and ensure that the IdM server that you want to configure is listed in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

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

3. Make a copy of the **ensure-SRV-record-is-present.yml** Ansible playbook file. For example:

```
$ cp ensure-SRV-record-is-present.yml ensure-SRV-record-is-present-copy.yml
```

4. Open the **ensure-SRV-record-is-present-copy.yml** file for editing.
5. Adapt the file by setting the following variables in the **ipadnsrecord** task section:

- Set the **ipaadmin_password** variable to your IdM administrator password.
- Set the **name** variable to **_kerberos._udp.idm.example.com**.
- Set the **srv_rec** variable to **'10 50 88 idm.example.com'**.
- Set the **zone_name** variable to **idm.example.com**.

This the modified Ansible playbook file for the current example:

```
---
- name: Test multiple DNS Records are present.
  hosts: ipaserver
  become: true
  gather_facts: false

  tasks:
    # Ensure a SRV record is present
    - ipadnsrecord:
        ipaadmin_password: Secret123
        name: _kerberos._udp.idm.example.com
        srv_rec: '10 50 88 idm.example.com'
        zone_name: idm.example.com
        state: present
```

6. Save the file.
7. Run the playbook:

```
$ ansible-playbook -v -i inventory.file ensure-SRV-record-is-present.yml
```


Additional resources

- For more information on SRV records, see [DNS records in IdM](#).
- You can see more sample Ansible playbooks for the **ansible-freeipa ipadnsrecord** module in the **README-dnsrecord.md** Markdown file available in the `/usr/share/doc/ansible-freeipa/` directory. The file also contains the definitions of the **ipadnsrecord** variables.
- You can see sample Ansible playbooks for the **ipadnsrecord** module in the `/usr/share/doc/ansible-freeipa/playbooks/dnsrecord` directory.