



Red Hat Enterprise Linux 9

Using Ansible to install and manage Identity Management

Using Ansible to maintain an IdM environment

Red Hat Enterprise Linux 9 Using Ansible to install and manage Identity Management

Using Ansible to maintain an IdM environment

Legal Notice

Copyright © 2025 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

Red Hat provides the `ansible-freeipa` package to enable administrators to run Red Hat Identity Management (IdM) by using Ansible. You can use playbooks to install IdM and manage users, groups, hosts, access control, and configuration settings.

Table of Contents

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. INSTALLING THE ANSIBLE-FREEIPA PACKAGE	11
2.3. ANSIBLE ROLES LOCATION IN THE FILE SYSTEM	12
2.4. SETTING THE PARAMETERS FOR A DEPLOYMENT WITH AN INTEGRATED DNS AND AN INTEGRATED CA AS THE ROOT CA	13
2.5. SETTING THE PARAMETERS FOR A DEPLOYMENT WITH EXTERNAL DNS AND AN INTEGRATED CA AS THE ROOT CA	16
2.6. DEPLOYING AN IDM SERVER WITH AN INTEGRATED CA AS THE ROOT CA USING AN ANSIBLE PLAYBOOK	18
2.7. SETTING THE PARAMETERS FOR A DEPLOYMENT WITH AN INTEGRATED DNS AND AN EXTERNAL CA AS THE ROOT CA	19
2.8. SETTING THE PARAMETERS FOR A DEPLOYMENT WITH EXTERNAL DNS AND AN EXTERNAL CA AS THE ROOT CA	22
2.9. DEPLOYING AN IDM SERVER WITH AN EXTERNAL CA AS THE ROOT CA USING AN ANSIBLE PLAYBOOK	25
2.10. UNINSTALLING AN IDM SERVER USING AN ANSIBLE PLAYBOOK	26
2.11. USING AN ANSIBLE PLAYBOOK TO UNINSTALL AN IDM SERVER EVEN IF THIS LEADS TO A DISCONNECTED TOPOLOGY	27
CHAPTER 3. INSTALLING AN IDENTITY MANAGEMENT REPLICA USING AN ANSIBLE PLAYBOOK	30
3.1. SPECIFYING THE BASE, SERVER AND CLIENT VARIABLES FOR INSTALLING THE IDM REPLICA	30
3.2. SPECIFYING THE CREDENTIALS FOR INSTALLING THE IDM REPLICA USING AN ANSIBLE PLAYBOOK	34
3.3. DEPLOYING AN IDM REPLICA USING AN ANSIBLE PLAYBOOK	35
3.4. UNINSTALLING AN IDM REPLICA USING AN ANSIBLE PLAYBOOK	36
CHAPTER 4. INSTALLING AN IDENTITY MANAGEMENT CLIENT USING AN ANSIBLE PLAYBOOK	37
4.1. SETTING THE PARAMETERS OF THE INVENTORY FILE FOR THE AUTODISCOVERY CLIENT INSTALLATION MODE	37
4.2. SETTING THE PARAMETERS OF THE INVENTORY FILE WHEN AUTODISCOVERY IS NOT POSSIBLE DURING CLIENT INSTALLATION	39
4.3. AUTHORIZATION OPTIONS FOR IDM CLIENT ENROLLMENT USING AN ANSIBLE PLAYBOOK	42
4.4. DEPLOYING AN IDM CLIENT USING AN ANSIBLE PLAYBOOK	43
4.5. USING THE ONE-TIME PASSWORD METHOD IN ANSIBLE TO INSTALL AN IDM CLIENT	44
4.6. TESTING AN IDENTITY MANAGEMENT CLIENT AFTER ANSIBLE INSTALLATION	46
4.7. UNINSTALLING AN IDM CLIENT USING AN ANSIBLE PLAYBOOK	46
CHAPTER 5. PREPARING YOUR ENVIRONMENT FOR MANAGING IDM USING ANSIBLE PLAYBOOKS	48
5.1. PREPARING A CONTROL NODE AND MANAGED NODES FOR MANAGING IDM USING ANSIBLE PLAYBOOKS	48
5.2. DIFFERENT METHODS TO PROVIDE THE CREDENTIALS REQUIRED FOR ANSIBLE-FREEIPA PLAYBOOKS	50
5.3. THE INVENTORY PLUGIN IN ANSIBLE-FREEIPA	52
CHAPTER 6. CONFIGURING GLOBAL IDM SETTINGS USING ANSIBLE PLAYBOOKS	54
6.1. RETRIEVING IDM CONFIGURATION USING AN ANSIBLE PLAYBOOK	54
6.2. CONFIGURING THE IDM CA RENEWAL SERVER USING AN ANSIBLE PLAYBOOK	56
6.3. CONFIGURING THE DEFAULT SHELL FOR IDM USERS USING AN ANSIBLE PLAYBOOK	57

6.4. CONFIGURING A NETBIOS NAME FOR AN IDM DOMAIN BY USING ANSIBLE	59
6.5. ENSURING THAT IDM USERS AND GROUPS HAVE SIDS BY USING ANSIBLE	60
6.6. ADDITIONAL RESOURCES	62
CHAPTER 7. MANAGING USER ACCOUNTS USING ANSIBLE PLAYBOOKS	63
7.1. USER LIFE CYCLE	63
7.2. ENSURING THE PRESENCE OF AN IDM USER USING AN ANSIBLE PLAYBOOK	64
7.3. ENSURING THE PRESENCE OF MULTIPLE IDM USERS USING ANSIBLE PLAYBOOKS	66
7.4. ENSURING THE PRESENCE OF MULTIPLE IDM USERS FROM A JSON FILE USING ANSIBLE PLAYBOOKS	68
7.5. ENSURING THE ABSENCE OF USERS USING ANSIBLE PLAYBOOKS	69
7.6. ADDITIONAL RESOURCES	71
CHAPTER 8. MANAGING USER GROUPS USING ANSIBLE PLAYBOOKS	72
8.1. THE DIFFERENT GROUP TYPES IN IDM	72
8.2. DIRECT AND INDIRECT GROUP MEMBERS	73
8.3. ENSURING THE PRESENCE OF IDM GROUPS AND GROUP MEMBERS USING ANSIBLE PLAYBOOKS	73
8.4. USING ANSIBLE TO ADD MULTIPLE IDM GROUPS IN A SINGLE TASK	75
8.5. USING ANSIBLE TO ENABLE AD USERS TO ADMINISTER IDM	76
8.6. ENSURING THE PRESENCE OF MEMBER MANAGERS IN IDM USER GROUPS USING ANSIBLE PLAYBOOKS	78
8.7. ENSURING THE ABSENCE OF MEMBER MANAGERS IN IDM USER GROUPS USING ANSIBLE PLAYBOOKS	80
CHAPTER 9. USING ANSIBLE TO AUTOMATE GROUP MEMBERSHIP IN IDM	82
9.1. USING ANSIBLE TO ENSURE THAT AN AUTOMEMBER RULE FOR AN IDM USER GROUP IS PRESENT	82
9.2. USING ANSIBLE TO ENSURE THAT A SPECIFIED CONDITION IS PRESENT IN AN IDM USER GROUP AUTOMEMBER RULE	83
9.3. USING ANSIBLE TO ENSURE THAT A CONDITION IS ABSENT FROM AN IDM USER GROUP AUTOMEMBER RULE	85
9.4. USING ANSIBLE TO ENSURE THAT AN AUTOMEMBER RULE FOR AN IDM USER GROUP IS ABSENT	87
9.5. USING ANSIBLE TO ENSURE THAT A CONDITION IS PRESENT IN AN IDM HOST GROUP AUTOMEMBER RULE	89
CHAPTER 10. USING ANSIBLE PLAYBOOKS TO MANAGE SELF-SERVICE RULES IN IDM	92
10.1. SELF-SERVICE ACCESS CONTROL IN IDM	92
10.2. USING ANSIBLE TO ENSURE THAT A SELF-SERVICE RULE IS PRESENT	92
10.3. USING ANSIBLE TO ENSURE THAT A SELF-SERVICE RULE IS ABSENT	94
10.4. USING ANSIBLE TO ENSURE THAT A SELF-SERVICE RULE HAS SPECIFIC ATTRIBUTES	95
10.5. USING ANSIBLE TO ENSURE THAT A SELF-SERVICE RULE DOES NOT HAVE SPECIFIC ATTRIBUTES	96
CHAPTER 11. DELEGATING PERMISSIONS TO USER GROUPS TO MANAGE USERS USING ANSIBLE PLAYBOOKS	99
11.1. DELEGATION RULES	99
11.2. CREATING AN ANSIBLE INVENTORY FILE FOR IDM	99
11.3. USING ANSIBLE TO ENSURE THAT A DELEGATION RULE IS PRESENT	100
11.4. USING ANSIBLE TO ENSURE THAT A DELEGATION RULE IS ABSENT	102
11.5. USING ANSIBLE TO ENSURE THAT A DELEGATION RULE HAS SPECIFIC ATTRIBUTES	103
11.6. USING ANSIBLE TO ENSURE THAT A DELEGATION RULE DOES NOT HAVE SPECIFIC ATTRIBUTES	105
CHAPTER 12. USING ANSIBLE PLAYBOOKS TO MANAGE ROLE-BASED ACCESS CONTROL IN IDM	107
12.1. PERMISSIONS IN IDM	107
12.2. DEFAULT MANAGED PERMISSIONS	108
12.3. PRIVILEGES IN IDM	109

12.4. ROLES IN IDM	110
12.5. PREDEFINED ROLES IN IDENTITY MANAGEMENT	110
12.6. USING ANSIBLE TO ENSURE AN IDM RBAC ROLE WITH PRIVILEGES IS PRESENT	111
12.7. USING ANSIBLE TO ENSURE AN IDM RBAC ROLE IS ABSENT	113
12.8. USING ANSIBLE TO ENSURE THAT A GROUP OF USERS IS ASSIGNED TO AN IDM RBAC ROLE	114
12.9. USING ANSIBLE TO ENSURE THAT SPECIFIC USERS ARE NOT ASSIGNED TO AN IDM RBAC ROLE	116
12.10. USING ANSIBLE TO ENSURE A SERVICE IS A MEMBER OF AN IDM RBAC ROLE	117
12.11. USING ANSIBLE TO ENSURE A HOST IS A MEMBER OF AN IDM RBAC ROLE	119
12.12. USING ANSIBLE TO ENSURE A HOST GROUP IS A MEMBER OF AN IDM RBAC ROLE	120
CHAPTER 13. USING ANSIBLE PLAYBOOKS TO MANAGE RBAC PRIVILEGES	123
13.1. USING ANSIBLE TO ENSURE A CUSTOM IDM RBAC PRIVILEGE IS PRESENT	123
13.2. USING ANSIBLE TO ENSURE MEMBER PERMISSIONS ARE PRESENT IN A CUSTOM IDM RBAC PRIVILEGE	124
13.3. USING ANSIBLE TO ENSURE AN IDM RBAC PRIVILEGE DOES NOT INCLUDE A PERMISSION	126
13.4. USING ANSIBLE TO RENAME A CUSTOM IDM RBAC PRIVILEGE	128
13.5. USING ANSIBLE TO ENSURE AN IDM RBAC PRIVILEGE IS ABSENT	129
13.6. ADDITIONAL RESOURCES	131
CHAPTER 14. USING ANSIBLE PLAYBOOKS TO MANAGE RBAC PERMISSIONS IN IDM	132
14.1. USING ANSIBLE TO ENSURE AN RBAC PERMISSION IS PRESENT	132
14.2. USING ANSIBLE TO ENSURE AN RBAC PERMISSION WITH AN ATTRIBUTE IS PRESENT	134
14.3. USING ANSIBLE TO ENSURE AN RBAC PERMISSION IS ABSENT	136
14.4. USING ANSIBLE TO ENSURE AN ATTRIBUTE IS A MEMBER OF AN IDM RBAC PERMISSION	137
14.5. USING ANSIBLE TO ENSURE AN ATTRIBUTE IS NOT A MEMBER OF AN IDM RBAC PERMISSION	139
14.6. USING ANSIBLE TO RENAME AN IDM RBAC PERMISSION	140
14.7. ADDITIONAL RESOURCES	141
CHAPTER 15. USING ANSIBLE TO MANAGE THE REPLICATION TOPOLOGY IN IDM	143
15.1. USING ANSIBLE TO ENSURE A REPLICATION AGREEMENT EXISTS IN IDM	143
15.2. USING ANSIBLE TO ENSURE REPLICATION AGREEMENTS EXIST BETWEEN MULTIPLE IDM REPLICAS	145
15.3. USING ANSIBLE TO CHECK IF A REPLICATION AGREEMENT EXISTS BETWEEN TWO REPLICAS	147
15.4. USING ANSIBLE TO VERIFY THAT A TOPOLOGY SUFFIX EXISTS IN IDM	149
15.5. USING ANSIBLE TO REINITIALIZE AN IDM REPLICA	150
15.6. USING ANSIBLE TO ENSURE A REPLICATION AGREEMENT IS ABSENT IN IDM	152
15.7. ADDITIONAL RESOURCES	154
CHAPTER 16. MANAGING IDM SERVERS BY USING ANSIBLE	155
16.1. CHECKING THAT AN IDM SERVER IS PRESENT BY USING ANSIBLE	155
16.2. ENSURING THAT AN IDM SERVER IS ABSENT FROM AN IDM TOPOLOGY BY USING ANSIBLE	156
16.3. ENSURING THE ABSENCE OF AN IDM SERVER DESPITE HOSTING A LAST IDM SERVER ROLE	158
16.4. ENSURING THAT AN IDM SERVER IS ABSENT BUT NOT NECESSARILY DISCONNECTED FROM OTHER IDM SERVERS	160
16.5. ENSURING THAT AN EXISTING IDM SERVER IS HIDDEN USING AN ANSIBLE PLAYBOOK	161
16.6. ENSURING THAT AN EXISTING IDM SERVER IS VISIBLE BY USING AN ANSIBLE PLAYBOOK	163
16.7. ENSURING THAT AN EXISTING IDM SERVER HAS AN IDM DNS LOCATION ASSIGNED	164
16.8. ENSURING THAT AN EXISTING IDM SERVER HAS NO IDM DNS LOCATION ASSIGNED	166
CHAPTER 17. MANAGING HOSTS USING ANSIBLE PLAYBOOKS	168
17.1. ENSURING THE PRESENCE OF AN IDM HOST ENTRY WITH FQDN USING ANSIBLE PLAYBOOKS	168
17.2. ENSURING THE PRESENCE OF AN IDM HOST ENTRY WITH DNS INFORMATION USING ANSIBLE PLAYBOOKS	169
17.3. ENSURING THE PRESENCE OF MULTIPLE IDM HOST ENTRIES WITH RANDOM PASSWORDS USING ANSIBLE PLAYBOOKS	171

17.4. ENSURING THE PRESENCE OF AN IDM HOST ENTRY WITH MULTIPLE IP ADDRESSES USING ANSIBLE PLAYBOOKS	173
17.5. ENSURING THE ABSENCE OF AN IDM HOST ENTRY USING ANSIBLE PLAYBOOKS	175
17.6. ADDITIONAL RESOURCES	177
CHAPTER 18. MANAGING HOST GROUPS USING ANSIBLE PLAYBOOKS	178
18.1. HOST GROUPS IN IDM	178
18.2. ENSURING THE PRESENCE OF IDM HOST GROUPS USING ANSIBLE PLAYBOOKS	178
18.3. ENSURING THE PRESENCE OF HOSTS IN IDM HOST GROUPS USING ANSIBLE PLAYBOOKS	180
18.4. NESTING IDM HOST GROUPS USING ANSIBLE PLAYBOOKS	181
18.5. ENSURING THE PRESENCE OF MEMBER MANAGERS IN IDM HOST GROUPS USING ANSIBLE PLAYBOOKS	183
18.6. ENSURING THE ABSENCE OF HOSTS FROM IDM HOST GROUPS USING ANSIBLE PLAYBOOKS	185
18.7. ENSURING THE ABSENCE OF NESTED HOST GROUPS FROM IDM HOST GROUPS USING ANSIBLE PLAYBOOKS	186
18.8. ENSURING THE ABSENCE OF IDM HOST GROUPS USING ANSIBLE PLAYBOOKS	188
18.9. ENSURING THE ABSENCE OF MEMBER MANAGERS FROM IDM HOST GROUPS USING ANSIBLE PLAYBOOKS	190
CHAPTER 19. DEFINING IDM PASSWORD POLICIES	192
19.1. WHAT IS A PASSWORD POLICY	192
19.2. PASSWORD POLICIES IN IDM	192
19.3. PASSWORD POLICY PRIORITIES IN IDM	194
19.4. ENSURING THE PRESENCE OF A PASSWORD POLICY IN IDM USING AN ANSIBLE PLAYBOOK	194
19.5. ADDING A NEW PASSWORD POLICY IN IDM USING THE WEBUI OR THE CLI	196
19.5.1. Adding a new password policy in the IdM WebUI	196
19.5.2. Adding a new password policy in the IdM CLI	197
19.6. ADDITIONAL PASSWORD POLICY OPTIONS IN IDM	197
19.7. APPLYING ADDITIONAL PASSWORD POLICY OPTIONS TO AN IDM GROUP	198
19.8. USING AN ANSIBLE PLAYBOOK TO APPLY ADDITIONAL PASSWORD POLICY OPTIONS TO AN IDM GROUP	200
CHAPTER 20. GRANTING SUDO ACCESS TO AN IDM USER ON AN IDM CLIENT	204
20.1. SUDO ACCESS ON AN IDM CLIENT	204
20.2. GRANTING SUDO ACCESS TO AN IDM USER ON AN IDM CLIENT USING THE CLI	204
20.3. GRANTING SUDO ACCESS TO AN AD USER ON AN IDM CLIENT USING THE CLI	207
20.4. GRANTING SUDO ACCESS TO AN IDM USER ON AN IDM CLIENT USING THE IDM WEB UI	210
20.5. CREATING A SUDO RULE ON THE CLI THAT RUNS A COMMAND AS A SERVICE ACCOUNT ON AN IDM CLIENT	212
20.6. CREATING A SUDO RULE IN THE IDM WEBUI THAT RUNS A COMMAND AS A SERVICE ACCOUNT ON AN IDM CLIENT	215
20.7. ENABLING GSSAPI AUTHENTICATION FOR SUDO ON AN IDM CLIENT	217
20.8. ENABLING GSSAPI AUTHENTICATION AND ENFORCING KERBEROS AUTHENTICATION INDICATORS FOR SUDO ON AN IDM CLIENT	219
20.9. SSSD OPTIONS CONTROLLING GSSAPI AUTHENTICATION FOR PAM SERVICES	222
20.10. TROUBLESHOOTING GSSAPI AUTHENTICATION FOR SUDO	223
20.11. USING AN ANSIBLE PLAYBOOK TO ENSURE SUDO ACCESS FOR AN IDM USER ON AN IDM CLIENT	225
20.12. MANAGING MULTIPLE IDM SUDO RULES IN A SINGLE ANSIBLE TASK	227
CHAPTER 21. ENSURING THE PRESENCE OF HOST-BASED ACCESS CONTROL RULES IN IDM USING ANSIBLE PLAYBOOKS	230
21.1. HOST-BASED ACCESS CONTROL RULES IN IDM	230
21.2. ENSURING THE PRESENCE OF AN HBAC RULE IN IDM USING AN ANSIBLE PLAYBOOK	230
CHAPTER 22. MANAGING IDM CERTIFICATES USING ANSIBLE	233

22.1. USING ANSIBLE TO REQUEST SSL CERTIFICATES FOR IDM HOSTS, SERVICES AND USERS	233
22.2. USING ANSIBLE TO REVOKE SSL CERTIFICATES FOR IDM HOSTS, SERVICES AND USERS	234
22.3. USING ANSIBLE TO RESTORE SSL CERTIFICATES FOR IDM USERS, HOSTS, AND SERVICES	235
22.4. USING ANSIBLE TO RETRIEVE SSL CERTIFICATES FOR IDM USERS, HOSTS, AND SERVICES	236
CHAPTER 23. VAULTS IN IDM	238
23.1. VAULTS AND THEIR BENEFITS	238
23.2. VAULT OWNERS, MEMBERS, AND ADMINISTRATORS	238
23.3. STANDARD, SYMMETRIC, AND ASYMMETRIC VAULTS	239
23.4. USER, SERVICE, AND SHARED VAULTS	240
23.5. VAULT CONTAINERS	240
23.6. BASIC IDM VAULT COMMANDS	240
23.7. INSTALLING THE KEY RECOVERY AUTHORITY IN IDM	241
CHAPTER 24. USING ANSIBLE TO MANAGE IDM USER VAULTS: STORING AND RETRIEVING SECRETS	243
24.1. ENSURING THE PRESENCE OF A STANDARD USER VAULT IN IDM USING ANSIBLE	243
24.2. ARCHIVING A SECRET IN A STANDARD USER VAULT IN IDM USING ANSIBLE	244
24.3. RETRIEVING A SECRET FROM A STANDARD USER VAULT IN IDM USING ANSIBLE	246
CHAPTER 25. USING ANSIBLE TO MANAGE IDM SERVICE VAULTS: STORING AND RETRIEVING SECRETS	249
25.1. ENSURING THE PRESENCE OF AN ASYMMETRIC SERVICE VAULT IN IDM USING ANSIBLE	249
25.2. ADDING MEMBER SERVICES TO AN ASYMMETRIC VAULT USING ANSIBLE	251
25.3. STORING AN IDM SERVICE SECRET IN AN ASYMMETRIC VAULT USING ANSIBLE	253
25.4. RETRIEVING A SERVICE SECRET FOR AN IDM SERVICE USING ANSIBLE	254
25.5. CHANGING AN IDM SERVICE VAULT SECRET WHEN COMPROMISED USING ANSIBLE	257
25.6. ADDITIONAL RESOURCES	260
CHAPTER 26. ENSURING THE PRESENCE AND ABSENCE OF SERVICES IN IDM USING ANSIBLE	261
26.1. ENSURING THE PRESENCE OF AN HTTP SERVICE IN IDM USING AN ANSIBLE PLAYBOOK	261
26.2. ENSURING THE PRESENCE OF MULTIPLE SERVICES IN IDM ON AN IDM CLIENT USING A SINGLE ANSIBLE TASK	263
26.3. ENSURING THE PRESENCE OF AN HTTP SERVICE IN IDM ON A NON-IDM CLIENT USING AN ANSIBLE PLAYBOOK	264
26.4. ENSURING THE PRESENCE OF AN HTTP SERVICE ON AN IDM CLIENT WITHOUT DNS USING AN ANSIBLE PLAYBOOK	265
26.5. ENSURING THE PRESENCE OF AN EXTERNALLY SIGNED CERTIFICATE IN AN IDM SERVICE ENTRY USING AN ANSIBLE PLAYBOOK	267
26.6. USING AN ANSIBLE PLAYBOOK TO ALLOW IDM USERS, GROUPS, HOSTS, OR HOST GROUPS TO CREATE A KEYTAB OF A SERVICE	269
26.7. USING AN ANSIBLE PLAYBOOK TO ALLOW IDM USERS, GROUPS, HOSTS, OR HOST GROUPS TO RETRIEVE A KEYTAB OF A SERVICE	271
26.8. ENSURING THE PRESENCE OF A KERBEROS PRINCIPAL ALIAS OF A SERVICE USING AN ANSIBLE PLAYBOOK	273
26.9. ENSURING THE ABSENCE OF AN HTTP SERVICE IN IDM USING AN ANSIBLE PLAYBOOK	275
26.10. ADDITIONAL RESOURCES	277
CHAPTER 27. MANAGING GLOBAL DNS CONFIGURATION IN IDM USING ANSIBLE PLAYBOOKS	278
27.1. HOW IDM ENSURES THAT GLOBAL FORWARDERS FROM /ETC/RESOLV.CONF ARE NOT REMOVED BY NETWORKMANAGER	278
27.2. ENSURING THE PRESENCE OF A DNS GLOBAL FORWARDER IN IDM USING ANSIBLE	279
27.3. ENSURING THE ABSENCE OF A DNS GLOBAL FORWARDER IN IDM USING ANSIBLE	281
27.4. THE ACTION: MEMBER OPTION IN IPADNSCONFIG ANSIBLE-FREEIPA MODULES	283
27.5. DNS FORWARD POLICIES IN IDM	284
27.6. USING AN ANSIBLE PLAYBOOK TO ENSURE THAT THE FORWARD FIRST POLICY IS SET IN IDM DNS GLOBAL CONFIGURATION	285

27.7. USING AN ANSIBLE PLAYBOOK TO ENSURE THAT GLOBAL FORWARDERS ARE DISABLED IN IDM DNS	287
27.8. USING AN ANSIBLE PLAYBOOK TO ENSURE THAT SYNCHRONIZATION OF FORWARD AND REVERSE LOOKUP ZONES IS DISABLED IN IDM DNS	288
CHAPTER 28. USING ANSIBLE PLAYBOOKS TO MANAGE IDM DNS ZONES	290
28.1. SUPPORTED DNS ZONE TYPES	290
28.2. CONFIGURATION ATTRIBUTES OF PRIMARY IDM DNS ZONES	291
28.3. USING ANSIBLE TO CREATE A PRIMARY ZONE IN IDM DNS	293
28.4. USING AN ANSIBLE PLAYBOOK TO ENSURE THE PRESENCE OF A PRIMARY DNS ZONE IN IDM WITH MULTIPLE VARIABLES	295
28.5. USING AN ANSIBLE PLAYBOOK TO ENSURE THE PRESENCE OF A ZONE FOR REVERSE DNS LOOKUP WHEN AN IP ADDRESS IS GIVEN	297
CHAPTER 29. USING ANSIBLE TO MANAGE DNS LOCATIONS IN IDM	300
29.1. DNS-BASED SERVICE DISCOVERY	300
29.2. DEPLOYMENT CONSIDERATIONS FOR DNS LOCATIONS	301
29.3. DNS TIME TO LIVE (TTL)	301
29.4. USING ANSIBLE TO ENSURE AN IDM LOCATION IS PRESENT	301
29.5. USING ANSIBLE TO ENSURE AN IDM LOCATION IS ABSENT	303
29.6. ADDITIONAL RESOURCES	304
CHAPTER 30. MANAGING DNS FORWARDING IN IDM	305
30.1. THE TWO ROLES OF AN IDM DNS SERVER	305
30.2. DNS FORWARD POLICIES IN IDM	306
30.3. ADDING A GLOBAL FORWARDER IN THE IDM WEB UI	306
30.4. ADDING A GLOBAL FORWARDER IN THE CLI	309
30.5. ADDING A DNS FORWARD ZONE IN THE IDM WEB UI	310
30.6. ADDING A DNS FORWARD ZONE IN THE CLI	313
30.7. ESTABLISHING A DNS GLOBAL FORWARDER IN IDM USING ANSIBLE	314
30.8. ENSURING THE PRESENCE OF A DNS GLOBAL FORWARDER IN IDM USING ANSIBLE	316
30.9. ENSURING THE ABSENCE OF A DNS GLOBAL FORWARDER IN IDM USING ANSIBLE	317
30.10. ENSURING DNS GLOBAL FORWARDERS ARE DISABLED IN IDM USING ANSIBLE	319
30.11. ENSURING THE PRESENCE OF A DNS FORWARD ZONE IN IDM USING ANSIBLE	320
30.12. ENSURING A DNS FORWARD ZONE HAS MULTIPLE FORWARDERS IN IDM USING ANSIBLE	322
30.13. ENSURING A DNS FORWARD ZONE IS DISABLED IN IDM USING ANSIBLE	324
30.14. ENSURING THE ABSENCE OF A DNS FORWARD ZONE IN IDM USING ANSIBLE	326
CHAPTER 31. USING ANSIBLE TO MANAGE DNS RECORDS IN IDM	328
31.1. DNS RECORDS IN IDM	328
31.2. COMMON IPA DNSRECORD-* OPTIONS	329
31.3. ENSURING THE PRESENCE OF A AND AAAA DNS RECORDS IN IDM USING ANSIBLE	331
31.4. ENSURING THE PRESENCE OF A AND PTR DNS RECORDS IN IDM USING ANSIBLE	333
31.5. ENSURING THE PRESENCE OF MULTIPLE DNS RECORDS IN IDM USING ANSIBLE	335
31.6. ENSURING THE PRESENCE OF MULTIPLE CNAME RECORDS IN IDM USING ANSIBLE	337
31.7. ENSURING THE PRESENCE OF AN SRV RECORD IN IDM USING ANSIBLE	339
CHAPTER 32. USING ANSIBLE TO AUTOMOUNT NFS SHARES FOR IDM USERS	341
32.1. AUTOFS AND AUTOMOUNT IN IDM	341
32.2. SETTING UP AN NFS SERVER WITH KERBEROS IN A RED HAT ENTERPRISE LINUX IDENTITY MANAGEMENT DOMAIN	342
32.3. CONFIGURING AUTOMOUNT LOCATIONS, MAPS, AND KEYS IN IDM BY USING ANSIBLE	343
32.4. USING ANSIBLE TO ADD IDM USERS TO A GROUP THAT OWNS NFS SHARES	345
32.5. CONFIGURING AUTOMOUNT ON AN IDM CLIENT	347
32.6. VERIFYING THAT AN IDM USER CAN ACCESS NFS SHARES ON AN IDM CLIENT	347

CHAPTER 33. USING ANSIBLE TO INTEGRATE IDM WITH NIS DOMAINS AND NETGROUPS	349
33.1. NIS AND ITS BENEFITS	349
33.2. NIS IN IDM	349
33.3. NIS NETGROUPS IN IDM	350
33.4. USING ANSIBLE TO ENSURE THAT A NETGROUP IS PRESENT	350
33.5. USING ANSIBLE TO ENSURE THAT MEMBERS ARE PRESENT IN A NETGROUP	351
33.6. USING ANSIBLE TO ENSURE THAT A MEMBER IS ABSENT FROM A NETGROUP	352
33.7. USING ANSIBLE TO ENSURE THAT A NETGROUP IS ABSENT	353
CHAPTER 34. USING ANSIBLE TO CONFIGURE HBAC AND SUDO RULES IN IDM	355
CHAPTER 35. USING ANSIBLE TO DELEGATE AUTHENTICATION FOR IDM USERS TO EXTERNAL IDENTITY PROVIDERS	360
35.1. THE BENEFITS OF CONNECTING IDM TO AN EXTERNAL IDP	360
35.2. HOW IDM INCORPORATES LOGINS VIA EXTERNAL IDPS	360
35.3. USING ANSIBLE TO CREATE A REFERENCE TO AN EXTERNAL IDENTITY PROVIDER	361
35.4. USING ANSIBLE TO ENABLE AN IDM USER TO AUTHENTICATE VIA AN EXTERNAL IDP	362
35.5. RETRIEVING AN IDM TICKET-GRANTING TICKET AS AN EXTERNAL IDP USER	364
35.6. LOGGING IN TO AN IDM CLIENT VIA SSH AS AN EXTERNAL IDP USER	366
35.7. THE PROVIDER OPTION IN THE IPAIDP ANSIBLE MODULE	366
CHAPTER 36. USING CONSTRAINED DELEGATION IN IDM	371
36.1. CONSTRAINED DELEGATION IN IDENTITY MANAGEMENT	371
36.2. CONFIGURING THE WEB CONSOLE TO ALLOW A USER AUTHENTICATED WITH A SMART CARD TO SSH TO A REMOTE HOST WITHOUT BEING ASKED TO AUTHENTICATE AGAIN	372
36.3. USING ANSIBLE TO CONFIGURE THE WEB CONSOLE TO ALLOW A USER AUTHENTICATED WITH A SMART CARD TO SSH TO A REMOTE HOST WITHOUT BEING ASKED TO AUTHENTICATE AGAIN	373
36.4. CONFIGURING A WEB CONSOLE TO ALLOW A USER AUTHENTICATED WITH A SMART CARD TO RUN SUDO WITHOUT BEING ASKED TO AUTHENTICATE AGAIN	376
36.5. USING ANSIBLE TO CONFIGURE A WEB CONSOLE TO ALLOW A USER AUTHENTICATED WITH A SMART CARD TO RUN SUDO WITHOUT BEING ASKED TO AUTHENTICATE AGAIN	377
36.6. ADDITIONAL RESOURCES	380
CHAPTER 37. JOINING RHEL SYSTEMS TO AN ACTIVE DIRECTORY BY USING RHEL SYSTEM ROLES	381
37.1. JOINING RHEL TO AN ACTIVE DIRECTORY DOMAIN BY USING THE AD_INTEGRATION RHEL SYSTEM ROLE	381

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our documentation. Let us know how we can improve it.

Submitting feedback through Jira (account required)

1. Log in to the [Jira](#) website.
2. Click **Create** in the top navigation bar
3. Enter a descriptive title in the **Summary** field.
4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
5. Click **Create** at the bottom of the dialogue.

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

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.

Prerequisites

- You understand [Ansible](#) and IdM concepts:
 - Ansible roles
 - Ansible nodes
 - Ansible inventory
 - Ansible tasks
 - Ansible modules
 - Ansible plays and playbooks

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.

Additional resources

- [Automating Red Hat Enterprise Linux Identity Management installation](#)
- [Planning Identity Management](#)
- [Preparing the system for IdM server installation](#)

2.2. INSTALLING THE ANSIBLE-FREEIPA PACKAGE

The following procedure describes how to install the the **ansible-freeipa** roles.

Prerequisites

- 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 rhel-9-for-x86_64-appstream-rpms
```

2. Install the IdM Ansible roles:

```
# dnf install ansible-freeipa
```

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

2.3. 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 license 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.

```
[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

```

2.4. SETTING THE PARAMETERS FOR A DEPLOYMENT WITH AN INTEGRATED DNS AND 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 in an environment that uses the IdM integrated DNS solution.



NOTE

The inventory in this procedure uses the **INI** format. You can, alternatively, use the **YAML** or **JSON** formats.

Procedure

1. Create a **~/MyPlaybooks/** directory:

```
$ mkdir MyPlaybooks
```

2. Create a **~/MyPlaybooks/inventory** file.
3. 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.
4. Specify the IdM domain and realm information.
5. Specify that you want to use integrated DNS by adding the following option:

```
ipaserver_setup_dns=true
```

6. Specify the DNS forwarding settings. Choose one of the following options:
 - Use the **ipaserver_auto_forwarders=true** option if you want the installer to use forwarders from the **/etc/resolv.conf** file. Do not use this option if the nameserver specified in the **/etc/resolv.conf** file is the localhost 127.0.0.1 address or if you are on a virtual private network and the DNS servers you are using are normally unreachable from the public internet.
 - Use the **ipaserver_forwarders** option to specify your forwarders manually. The installation process adds the forwarder IP addresses to the **/etc/named.conf** file on the installed IdM server.
 - Use the **ipaserver_no_forwarders=true** option to configure root DNS servers to be used instead.

**NOTE**

With no DNS forwarders, your environment is isolated, and names from other DNS domains in your infrastructure are not resolved.

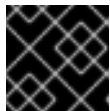
7. Specify the DNS reverse record and zone settings. Choose from the following options:

- Use the **ipaserver_allow_zone_overlap=true** option to allow the creation of a (reverse) zone even if the zone is already resolvable.
- Use the **ipaserver_reverse_zones** option to specify your reverse zones manually.
- Use the **ipaserver_no_reverse=true** option if you do not want the installer to create a reverse DNS zone.

**NOTE**

Using IdM to manage reverse zones is optional. You can use an external DNS service for this purpose instead.

8. 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.
9. 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=true
ipaserver_auto_forwarders=true
[...]
```

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=true
ipaserver_auto_forwarders=true
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=true
ipaserver_auto_forwarders=true
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

```

Additional resources

- `man ipa-server-install(1)`
- `/usr/share/doc/ansible-freeipa/README-server.md`

2.5. SETTING THE PARAMETERS FOR A DEPLOYMENT WITH EXTERNAL DNS AND 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 in an environment that uses an external DNS solution.



NOTE

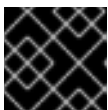
The inventory file in this procedure uses the **INI** format. You can, alternatively, use the **YAML** or **JSON** formats.

Procedure

1. Create a `~/MyPlaybooks/` directory:

```
$ mkdir MyPlaybooks
```

2. Create a `~/MyPlaybooks/inventory` file.
3. 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.
4. Specify the IdM domain and realm information.
5. Make sure that the `ipaserver_setup_dns` option is set to **no** or that it is absent.
6. 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.
7. 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=no
[...]
```

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=no
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=no
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
```

Additional resources

- `man ipa-server-install(1)`
- `/usr/share/doc/ansible-freeipa/README-server.md`

2.6. 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.

Prerequisites

- The managed node is a Red Hat Enterprise Linux 9 system with a static IP address and a working package manager.
- You have set the parameters that correspond to your scenario by choosing one of the following procedures:
 - [Procedure with integrated DNS](#)
 - [Procedure with external DNS](#)

Procedure

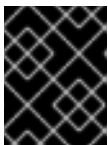
1. Run the Ansible playbook:

```
$ ansible-playbook -i ~/MyPlaybooks/inventory ~/MyPlaybooks/install-server.yml
```

2. Choose one of the following options:

- If your IdM deployment uses external DNS: add the DNS resource records contained in the `/tmp/ipa.system.records.UFRPto.db` file to the existing external DNS servers. The process of updating the DNS records varies depending on the particular DNS solution.

```
...
Restarting the KDC
Please add records in this file to your DNS system:
/tmp/ipa.system.records.UFRBto.db
Restarting the web server
...
```



IMPORTANT

The server installation is not complete until you add the DNS records to the existing DNS servers.

- If your IdM deployment uses integrated DNS:
 - Add DNS delegation from the parent domain to the IdM DNS domain. For example, if the IdM DNS domain is ***idm.example.com***, add a name server (NS) record to the ***example.com*** parent domain.

**IMPORTANT**

Repeat this step each time after an IdM DNS server is installed.

- Add an **_ntp._udp** service (SRV) record for your time server to your IdM DNS. The presence of the SRV record for the time server of the newly-installed IdM server in IdM DNS ensures that future replica and client installations are automatically configured to synchronize with the time server used by this primary IdM server.

2.7. SETTING THE PARAMETERS FOR A DEPLOYMENT WITH AN INTEGRATED DNS AND 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 in an environment that uses the IdM integrated DNS solution.

**NOTE**

The inventory file in this procedure uses the **INI** format. You can, alternatively, use the **YAML** or **JSON** formats.

Procedure

1. Create a **~/MyPlaybooks/** directory:

```
$ mkdir MyPlaybooks
```

2. Create a **~/MyPlaybooks/inventory** file.
3. 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.
4. Specify the IdM domain and realm information.
5. Specify that you want to use integrated DNS by adding the following option:

```
ipaserver_setup_dns=true
```

6. Specify the DNS forwarding settings. Choose one of the following options:
 - Use the **ipaserver_auto_forwarders=true** option if you want the installation process to use forwarders from the **/etc/resolv.conf** file. This option is not recommended if the nameserver specified in the **/etc/resolv.conf** file is the localhost 127.0.0.1 address or if you are on a virtual private network and the DNS servers you are using are normally unreachable from the public internet.
 - Use the **ipaserver_forwarders** option to specify your forwarders manually. The installation process adds the forwarder IP addresses to the **/etc/named.conf** file on the installed IdM server.

- Use the **ipaserver_no_forwarders=true** option to configure root DNS servers to be used instead.

**NOTE**

With no DNS forwarders, your environment is isolated, and names from other DNS domains in your infrastructure are not resolved.

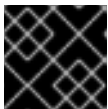
7. Specify the DNS reverse record and zone settings. Choose from the following options:

- Use the **ipaserver_allow_zone_overlap=true** option to allow the creation of a (reverse) zone even if the zone is already resolvable.
- Use the **ipaserver_reverse_zones** option to specify your reverse zones manually.
- Use the **ipaserver_no_reverse=true** option if you do not want the installation process to create a reverse DNS zone.

**NOTE**

Using IdM to manage reverse zones is optional. You can use an external DNS service for this purpose instead.

8. 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.
9. Optional: Specify a custom **firewalld** zone to be used by the IdM server. If you do not set a custom zone, IdM adds 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=true
ipaserver_auto_forwarders=true
[...]
```

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=true
ipaserver_auto_forwarders=true
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=true
ipaserver_auto_forwarders=true
ipaadmin_password=MySecretPassword123
ipadm_password=MySecretPassword234
ipaserver_firewalld_zone=custom zone

[...]
```

10. 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: true

  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: true
```

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

```
---
- name: Playbook to configure IPA server Step 2
  hosts: ipaserver
```

```

become: true
vars_files:
- playbook_sensitive_data.yml
vars:
  ipaserver_external_cert_files:
    - "/root/servercert20240601.pem"
    - "/root/cacert.pem"

pre_tasks:
- name: Copy "{{ groups.ipaserver[0] }}-{{ item }}" to "/root/{{ item }}" on node
  ansible.builtin.copy:
    src: "{{ groups.ipaserver[0] }}-{{ item }}"
    dest: "/root/{{ item }}"
    force: true
  with_items:
    - servercert20240601.pem
    - cacert.pem

roles:
- role: ipaserver
  state: present

```

Additional resources

- `man ipa-server-install(1)`
- `/usr/share/doc/ansible-freeipa/README-server.md`

2.8. SETTING THE PARAMETERS FOR A DEPLOYMENT WITH EXTERNAL DNS AND 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 in an environment that uses an external DNS solution.



NOTE

The inventory file in this procedure uses the **INI** format. You can, alternatively, use the **YAML** or **JSON** formats.

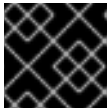
Procedure

1. Create a `~/MyPlaybooks/` directory:

```
$ mkdir MyPlaybooks
```

2. Create a `~/MyPlaybooks/inventory` file.
3. 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.

4. Specify the IdM domain and realm information.
5. Make sure that the **ipaserver_setup_dns** option is set to **no** or that it is absent.
6. 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.
7. 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=no
[...]
```

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=no
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=no
ipaadmin_password=MySecretPassword123
ipadm_password=MySecretPassword234
```

```
ipaserver_firewalld_zone=custom zone
```

```
[...]
```

8. 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: true

  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: true
```

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

```
---
- name: Playbook to configure IPA server Step 2
  hosts: ipaserver
  become: true
  vars_files:
    - playbook_sensitive_data.yml
  vars:
    ipaserver_external_cert_files:
      - "/root/servercert20240601.pem"
      - "/root/cacert.pem"

  pre_tasks:
    - name: Copy "{{ groups.ipaserver[0] }}-{{ item }}" to "/root/{{ item }}" on node
      ansible.builtin.copy:
        src: "{{ groups.ipaserver[0] }}-{{ item }}"
        dest: "/root/{{ item }}"
        force: true
      with_items:
        - servercert20240601.pem
        - cacert.pem

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

- [Installing an IdM server: Without integrated DNS, with an external CA as the root CA](#)
- `man ipa-server-install(1)`
- `/usr/share/doc/ansible-freeipa/README-server.md`

2.9. 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.

Prerequisites

- The managed node is a Red Hat Enterprise Linux 9 system with a static IP address and a working package manager.
- You have set the parameters that correspond to your scenario by choosing one of the following procedures:
 - [Procedure with integrated DNS](#)
 - [Procedure with external DNS](#)

Procedure

1. Run the Ansible playbook with the instructions for the first step of the installation, for example **install-server-step1.yml**:

```
$ ansible-playbook --vault-password-file=password_file -v -i ~/MyPlaybooks/inventory
~/MyPlaybooks/install-server-step1.yml
```

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 with the instructions for the final step of the installation, for example **install-server-step2.yml**:

```
$ ansible-playbook -v -i ~/MyPlaybooks/inventory ~/MyPlaybooks/install-server-
step2.yml
```

5. Choose one of the following options:
 - If your IdM deployment uses external DNS: add the DNS resource records contained in the **/tmp/ipa.system.records.UFRPto.db** file to the existing external DNS servers. The process of updating the DNS records varies depending on the particular DNS solution.

```
...
Restarting the KDC
Please add records in this file to your DNS system:
```

/tmp/ipa.system.records.UFRBto.db

Restarting the web server

...



IMPORTANT

The server installation is not complete until you add the DNS records to the existing DNS servers.

- If your IdM deployment uses integrated DNS:
 - Add DNS delegation from the parent domain to the IdM DNS domain. For example, if the IdM DNS domain is **idm.example.com**, add a name server (NS) record to the **example.com** parent domain.

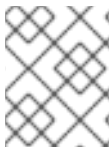


IMPORTANT

Repeat this step each time after an IdM DNS server is installed.

- Add an **_ntp._udp** service (SRV) record for your time server to your IdM DNS. The presence of the SRV record for the time server of the newly-installed IdM server in IdM DNS ensures that future replica and client installations are automatically configured to synchronize with the time server used by this primary IdM server.

2.10. UNINSTALLING AN IDM SERVER USING AN ANSIBLE PLAYBOOK



NOTE

In an existing Identity Management (IdM) deployment, **replica** and **server** are interchangeable terms.

Complete this procedure to uninstall an IdM replica using an Ansible playbook. In this example:

- IdM configuration is uninstalled from **server123.idm.example.com**.
- **server123.idm.example.com** and the associated host entry are removed from the IdM topology.

Prerequisites

- On the control node:
 - You are using Ansible version 2.14 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 in the **~/MyPlaybooks/** directory.
 - You have stored your **ipadmin_password** in the **secret.yml** Ansible vault.
 - For the **ipaserver_remove_from_topology** option to work, the system must be running on RHEL 9.3 or later.

- On the managed node:
 - The system is running on RHEL 9.

Procedure

1. Create your Ansible playbook file **uninstall-server.yml** with the following content:

```
---
- name: Playbook to uninstall an IdM replica
  hosts: ipaserver
  become: true

  roles:
  - role: ipaserver
    ipaserver_remove_from_domain: true
    state: absent
```

The **ipaserver_remove_from_domain** option unenrolls the host from the IdM topology.



NOTE

If the removal of **server123.idm.example.com** should lead to a disconnected topology, the removal will be aborted. For more information, see [Using an Ansible playbook to uninstall an IdM server even if this leads to a disconnected topology](#).

2. Uninstall the replica:

```
$ ansible-playbook --vault-password-file=password_file -v -i
<path_to_inventory_directory>/inventory <path_to_playbooks_directory>/uninstall-
server.yml
```

3. Ensure that all name server (NS) DNS records pointing to **server123.idm.example.com** are deleted from your DNS zones. This applies regardless of whether you use integrated DNS managed by IdM or external DNS. For more information on how to delete DNS records from IdM, see [Deleting DNS records in the IdM CLI](#).

2.11. USING AN ANSIBLE PLAYBOOK TO UNINSTALL AN IDM SERVER EVEN IF THIS LEADS TO A DISCONNECTED TOPOLOGY



NOTE

In an existing Identity Management (IdM) deployment, **replica** and **server** are interchangeable terms.

Complete this procedure to uninstall an IdM replica using an Ansible playbook even if this results in a disconnected IdM topology. In the example, **server456.idm.example.com** is used to remove the replica and the associated host entry with the FQDN of **server123.idm.example.com** from the topology, leaving certain replicas disconnected from **server456.idm.example.com** and the rest of the topology.



NOTE

If removing a replica from the topology using only the **remove_server_from_domain** does not result in a disconnected topology, no other options are required. If the result is a disconnected topology, you must specify which part of the domain you want to preserve. In that case, you must do the following:

- Specify the **ipaserver_remove_on_server** value.
- Set **ipaserver_ignore_topology_disconnect** to True.

Prerequisites

- On the control node:
 - You are using Ansible version 2.14 or later.
 - The system is running on RHEL 9.3 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 in the **~/MyPlaybooks/** directory.
 - You have stored your **ipaadmin_password** in the **secret.yml** Ansible vault.
- On the managed node:
 - The system is running on 9 or later.

Procedure

1. Create your Ansible playbook file **uninstall-server.yml** with the following content:

```
---
- name: Playbook to uninstall an IdM replica
  hosts: ipaserver
  become: true

  roles:
  - role: ipaserver
    ipaserver_remove_from_domain: true
    ipaserver_remove_on_server: server456.idm.example.com
    ipaserver_ignore_topology_disconnect: true
    state: absent
```



NOTE

Under normal circumstances, if the removal of server123 does not result in a disconnected topology: if the value for **ipaserver_remove_on_server** is not set, the replica on which server123 is removed is automatically determined using the replication agreements of server123.

2. Uninstall the replica:


```
$ ansible-playbook --vault-password-file=password_file -v -i  
<path_to_inventory_directory>/hosts <path_to_playbooks_directory>/uninstall-  
server.yml
```

3. Ensure that all name server (NS) DNS records pointing to **server123.idm.example.com** are deleted from your DNS zones. This applies regardless of whether you use integrated DNS managed by IdM or external DNS. For more information on how to delete DNS records from IdM, see [Deleting DNS records in the IdM CLI](#).

Additional resources

- [Inventory basics: formats, hosts, and groups](#)
- You can see sample Ansible playbooks for installing an IdM server and a list of possible variables in the [ansible-freeipa upstream documentation](#).

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

Configuring a system as an IdM replica by using [Ansible](#) 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.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible control node.
- You understand the general [Ansible](#) and IdM concepts.
- You have [planned the replica topology in your deployment](#) .

3.1. 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.

Prerequisites

- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package on the Ansible controller.

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.

- Optional: Provide additional information in the inventory file based on how you have designed your topology:

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
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
```

- Optional: Provide additional information regarding **firewalld** and DNS:

Scenario 1

If you want the replica to use a specified **firewalld** zone, for example an internal one, you can specify it in the inventory file. 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
```

Scenario 2

If you want the replica to host the IdM DNS service, add the **ipareplica_setup_dns=true** line to the **[ipareplicas:vars]** section. Additionally, specify if you want to use per-server DNS forwarders:

- To configure per-server forwarders, add the **ipareplica_forwarders** variable and a list of strings to the **[ipareplicas:vars]** section, for example:
ipareplica_forwarders=192.0.2.1,192.0.2.2
- To configure no per-server forwarders, add the following line to the **[ipareplicas:vars]** section: **ipareplica_no_forwarders=true**.
- To configure per-server forwarders based on the forwarders listed in the **/etc/resolv.conf** file of the replica, add the **ipareplica_auto_forwarders** variable to the **[ipareplicas:vars]** section.

Example inventory file with instructions to set up DNS and per-server forwarders on the replicas

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

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

[ipareplicas:vars]
ipareplica_setup_dns=true
ipareplica_forwarders=192.0.2.1,192.0.2.2
```

Scenario 3

Specify the DNS resolver using the **ipaclient_configure_dns_resolve** and **ipaclient_dns_servers** options (if available) to simplify cluster deployments. This is especially useful if your IdM deployment is using integrated DNS:

An inventory file snippet specifying a DNS resolver:

```
[...]
[ipaclient:vars]
ipaclient_configure_dns_resolver=true
ipaclient_dns_servers=192.168.100.1
```



NOTE

The **ipaclient_dns_servers** list must contain only IP addresses. Host names are not allowed.

Additional resources

- **/usr/share/ansible/roles/ipareplica/README.md**

3.2. SPECIFYING THE CREDENTIALS FOR INSTALLING THE IDM REPLICA USING AN ANSIBLE PLAYBOOK

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

Prerequisites

- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package on the Ansible controller.

Procedure

1. Specify the **password of a user authorized to deploy replicas** for example the IdM **admin**.
 - Use the Ansible Vault to store the password, and reference 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
```



NOTE

As of RHEL 9.5, during the installation of an IdM replica, checking if the provided Kerberos principal has the required privilege also extends to checking user ID overrides. As a result, you can deploy a replica using the credentials of an AD administrator that is configured to act as an IdM administrator.

Additional resources

- [/usr/share/ansible/roles/ipareplica/README.md](#)

3.3. DEPLOYING AN IDM REPLICA USING AN ANSIBLE PLAYBOOK

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

Prerequisites

- The managed node is a Red Hat Enterprise Linux 9 system with a static IP address and a working package manager.
- You have configured [the inventory file for installing an IdM replica](#) .
- You have configured [the authorization for installing the IdM replica](#) .

Procedure

- Run the Ansible playbook:

```
$ ansible-playbook -i ~/MyPlaybooks/inventory ~/MyPlaybooks/install-replica.yml
```

Next steps

- In large deployments, you might want to tune specific parameters of IdM replicas for better performance. Consult the [Tuning Performance in Identity Management](#) title to find tuning instructions to best suit your scenario.

3.4. UNINSTALLING AN IDM REPLICA USING AN ANSIBLE PLAYBOOK



NOTE

In an existing Identity Management (IdM) deployment, **replica** and **server** are interchangeable terms. For information on how to uninstall an IdM server, see [Uninstalling an IdM server using an Ansible playbook](#) or [Using an Ansible playbook to uninstall an IdM server even if this leads to a disconnected topology](#).

Additional resources

- [Introduction to IdM servers and clients](#)

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

Learn more about 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.

Prerequisites

- You have installed the [ansible-freeipa](#) package on the Ansible control node.
- You are using Ansible version 2.15 or later.
- You understand the general [Ansible](#) and IdM concepts.

4.1. SETTING THE PARAMETERS OF THE INVENTORY FILE FOR THE AUTODISCOVERY CLIENT INSTALLATION MODE

To install an Identity Management (IdM) client using an Ansible playbook, configure the target host parameters in an inventory file, for example **inventory**:

- 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_mkhome** variable in your Ansible playbook.

Procedure

1. Open your **inventory** file for editing.
2. 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
[...]
```

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.
 - Use the Ansible Vault to store the password, and reference 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: ipacient
      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: ipacient
      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 **[ipacient: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=true** option in your inventory file. For example, you can uncomment the **ipaclient_use_otp=true** 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**.
- 4. Optional: Specify the DNS resolver using the **ipaclient_configure_dns_resolve** and **ipaclient_dns_servers** options (if available) to simplify cluster deployments. This is especially useful if your IdM deployment is using integrated DNS:

An inventory file snippet specifying a DNS resolver:

```
[...]
[ipaclients:vars]
ipaadmin_password: "{{ ipaadmin_password }}"
ipaclient_domain=idm.example.com
ipaclient_configure_dns_resolver=true
ipaclient_dns_servers=192.168.100.1
```



NOTE

The **ipaclient_dns_servers** list must contain only IP addresses. Host names are not allowed.

- 5. Starting with RHEL 9.3, you can also specify the **ipaclient_subid: true** option to have subid ranges configured for IdM users on the IdM level.

Additional resources

- [/usr/share/ansible/roles/ipaclient/README.md](#)
- [Managing subID ranges manually](#)

4.2. 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, configure the target host parameters 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.
 - Use the Ansible Vault to store the password, and reference 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, the credentials of **admin** to be provided 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=true** option in your inventory file. For example, you can uncomment the **#ipaclient_use_otp=true** 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**.
4. Starting with RHEL 9.3, you can also specify the **ipaclient_subid: true** option to have subid ranges configured for IdM users on the IdM level.

Additional resources

- [/usr/share/ansible/roles/ipaclient/README.md](#)
- [Managing subID ranges manually](#)

4.3. AUTHORIZATION OPTIONS FOR IDM CLIENT ENROLLMENT USING AN ANSIBLE PLAYBOOK

You can authorize IdM client enrollment by using any of the following methods:

- A random, one-time password (OTP) + administrator password
- A random, one-time password (OTP) + an admin keytab
- The client keytab from the previous enrollment
- The password of a user authorized to enroll a client (**admin**) stored in an inventory file
- The password of a user authorized to enroll a client (**admin**) stored in an Ansible vault

It is possible to have the OTP generated by an IdM administrator before the IdM client installation. In that case, you do not need any credentials for the installation other than the OTP itself.

The following are sample inventory files for these methods:

Table 4.1. Sample inventory files

Authorization option	Inventory file
A random, one-time password (OTP) + administrator password	<pre>[ipaclients:vars] ipaadmin_password=Secret123 ipaclient_use_otp=true</pre>
A random, one-time password (OTP)	<pre>[ipaclients:vars] ipaclient_otp=<W5YpARl=7M.></pre> <p>This scenario assumes that the OTP was already generated by an IdM admin before the installation.</p>
A random, one-time password (OTP) + an admin keytab	<pre>[ipaclients:vars] ipaadmin_keytab=/root/admin.keytab ipaclient_use_otp=true</pre>
The client keytab from the previous enrollment	<pre>[ipaclients:vars] ipaclient_keytab=/root/krb5.keytab</pre>
Password of an admin user stored in an inventory file	<pre>[ipaclients:vars] ipaadmin_password=Secret123</pre>

Authorization option	Inventory file
Password of an admin user stored in an Ansible vault file	<pre>[ipaclients:vars] [...]</pre>

If you are using the password of an **admin** user stored in an Ansible vault file, the corresponding playbook file must have an additional **vars_files** directive:

Table 4.2. User password stored in an Ansible vault

Inventory file	Playbook file
<pre>[ipaclients:vars] [...]</pre>	<pre>- name: Playbook to configure IPA clients hosts: ipaclients become: true vars_files: - ansible_vault_file.yml roles: - role: ipaclient state: present</pre>

In all the other authorization scenarios described above, a basic playbook file could look as follows:

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

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



NOTE

As of RHEL 9.2, in the two OTP authorization scenarios described above, the requesting of the administrator's TGT by using the **kinit** command occurs on the first specified or discovered IdM server. Therefore, no additional modification of the Ansible control node is required. Before RHEL 9.2, the **krb5-workstation** package was required on the control node.

4.4. 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.

Prerequisites

- The managed node is a Red Hat Enterprise Linux 9 system with a static IP address and a working package manager.
- You have set 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 when autodiscovery is not possible during client installation](#)

Procedure

- Run the Ansible playbook:

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

4.5. USING THE ONE-TIME PASSWORD METHOD IN ANSIBLE TO INSTALL AN IDM CLIENT

You can generate a one-time password (OTP) for a new host in Identity Management (IdM) and use it to enroll a system into the IdM domain. This procedure describes how to use Ansible to install an IdM client after generating an OTP for it on another IdM host.

This method of installing an IdM client is convenient if two system administrators with different privileges exist in your organisation:

- One that has the credentials of an IdM administrator.
- Another that has the required Ansible credentials, including **root** access to the host to become an IdM client.

The IdM administrator performs the first part of the procedure in which the OTP password is generated. The Ansible administrator performs the remaining part of the procedure in which the OTP is used to install an IdM client.

Prerequisites

- You have the IdM **admin** credentials or at least the **Host Enrollment** privilege and a permission to add DNS records in IdM.
- You have configured a user escalation method on the Ansible managed node to allow you to install an IdM client.
- If your Ansible control node is running on RHEL 8.7 or earlier, you must be able to install packages on your Ansible control node.
- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package on the Ansible controller.
 - You have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.

- The managed node is a Red Hat Enterprise Linux 9 system with a static IP address and a working package manager.

Procedure

1. **SSH** to an IdM host as an IdM user with a role that has the **Host Enrollment** privilege and a permission to add DNS records:

```
$ ssh admin@server.idm.example.com
```

2. Generate an OTP for the new client:

```
[admin@server ~]$ ipa host-add client.idm.example.com --ip-address=172.25.250.11 --
random
-----
Added host "client.idm.example.com"
-----
Host name: client.idm.example.com
Random password: W5YpARl=7M.n
Password: True
Keytab: False
Managed by: server.idm.example.com
```

The `--ip-address=<your_host_ip_address>` option adds the host to IdM DNS with the specified IP address.

3. Exit the IdM host:

```
$ exit
logout
Connection to server.idm.example.com closed.
```

4. On the ansible controller, update the inventory file to include the random password:

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

[ipclients:vars]
ipaclient_domain=idm.example.com
ipaclient_otp=W5YpARl=7M.n
[...]
```

5. If your ansible controller is running RHEL \leq 9.1, install the **kinit** utility provided by the **krb5-workstation** package:

```
$ sudo dnf install krb5-workstation
```

6. Run the playbook to install the client:

```
$ ansible-playbook -i inventory install-client.yml
```

4.6. TESTING AN IDENTITY MANAGEMENT CLIENT AFTER ANSIBLE INSTALLATION

The command line (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 ~]$
```

4.7. 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.
- The managed node is a Red Hat Enterprise Linux 9 system with a static IP address.

Procedure

- Run the Ansible playbook with the instructions to uninstall the client, for example **uninstall-client.yml**:

```
$ ansible-playbook -v -i ~/MyPlaybooks/inventory ~/MyPlaybooks/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

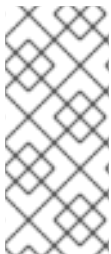
- [Uninstalling an IdM client](#)

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:

- Keep 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.



NOTE

You can run your **ansible-freeipa** playbooks without invoking **root** privileges on the managed nodes. Exceptions include playbooks that use the **ipaserver**, **ipareplica**, **ipaclient**, **ipasmartcard_server**, **ipasmartcard_client** and **ipabackup ansible-freeipa** roles. These roles require privileged access to directories and the **dnf** software package manager.

The playbooks in the Red Hat Enterprise Linux IdM documentation assume the following [security configuration](#):

- The IdM **admin** is your remote Ansible user on the managed nodes.
- You store the IdM **admin** password encrypted in an Ansible vault.
- You have placed the password that protects the Ansible vault in a password file.
- You block access to the vault password file to everyone except your local ansible user.
- You regularly remove and re-create the vault password file.

Consider also [alternative security configurations](#).

5.1. PREPARING A CONTROL NODE AND MANAGED NODES FOR MANAGING IDM USING ANSIBLE PLAYBOOKS

Follow this procedure 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. Change into the `~/MyPlaybooks/` directory:

```
$ cd ~/MyPlaybooks
```

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

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

3. 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.

4. 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
```

5. 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.

6. Create a **password_file** file that contains the vault password:

```
redhat
```

7. Change the permissions to modify the file:

```
$ chmod 0600 password_file
```

8. Create a **secret.yml** Ansible vault to store the IdM **admin** password:

- a. Configure **password_file** to store the vault password:

■

```
$ ansible-vault create --vault-password-file=password_file secret.yml
```

- b. When prompted, enter the content of the **secret.yml** file:

```
ipaadmin_password: Secret123
```

NOTE

To use the encrypted **ipaadmin_password** in a playbook, you must use the **vars_file** directive. For example, a simple playbook to delete an IdM user can look as follows:

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

  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml

  tasks:
    - name: Delete user robot
      ipauser:
        ipaadmin_password: "{{ ipaadmin_password }}"
        name: robot
        state: absent
```

When executing a playbook, instruct Ansible use the vault password to decrypt **ipaadmin_password** by adding the **--vault-password-file=password_file** option. For example:

```
ansible-playbook -i inventory --vault-password-file=password_file del-user.yml
```



WARNING

For security reasons, remove the vault password file at the end of each session, and repeat steps 6–8 at the start of each new session.

Additional resources

- [Different methods to provide the credentials required for ansible-freeipa playbooks](#)
- [Installing an Identity Management server using an Ansible playbook](#)
- [How to build your inventory](#)

5.2. DIFFERENT METHODS TO PROVIDE THE CREDENTIALS REQUIRED FOR ANSIBLE-FREEIPA PLAYBOOKS

There are advantages and disadvantages in the different methods for providing the credentials required for running playbooks that use **ansible-freeipa** roles and modules.

Storing passwords in plain text in a playbook

Benefits:

- Not being prompted all the time you run the playbook.
- Easy to implement.

Drawbacks:

- Everyone with access to the file can read the password. Setting wrong permissions and sharing the file, for example in an internal or external repository, can compromise security.
- High maintenance work: if the password is changed, it needs to be changed in all playbooks.

Entering passwords interactively when you execute a playbook

Benefits:

- No-one can steal the password as it is not stored anywhere.
- You can update the password easily.
- Easy to implement.

Drawbacks:

- If you are using Ansible playbooks in scripts, the requirement to enter the password interactively can be inconvenient.

Storing passwords in an Ansible vault and the vault password in a file:

Benefits:

- The user password is stored encrypted.
- You can update the user password easily, by creating a new Ansible vault.
- You can update the password file that protects the ansible vault easily, by using the **ansible-vault rekey --new-vault-password-file=NEW_VAULT_PASSWORD_FILE secret.yml** command.
- If you are using Ansible playbooks in scripts, it is convenient not to have to enter the password protecting the Ansible vault interactively.

Drawbacks:

- It is vital that the file that contains the sensitive plain text password be protected through file permissions and other security measures.

Storing passwords in an Ansible vault and entering the vault password interactively

Benefits:

- The user password is stored encrypted.

- No-one can steal the vault password as it is not stored anywhere.
- You can update the user password easily, by creating a new Ansible vault.
- You can update the vault password easily too, by using the **ansible-vault rekey *file_name*** command.

Drawbacks:

- If you are using Ansible playbooks in scripts, the need to enter the vault password interactively can be inconvenient.

Additional resources

- [Preparing a control node and managed nodes for managing IdM using Ansible playbooks](#)
- [What is Zero trust?](#)
- [Protecting sensitive data with Ansible vault](#)

5.3. THE INVENTORY PLUGIN IN ANSIBLE-FREEIPA

You can use the **ansible-freeipa freeipa** inventory plugin to create dynamic inventories of Identity Management (IdM) servers to be used in **ansible-freeipa** playbooks. The plugin gathers data about the IdM servers in the domain, and selects only those that have a specified IdM server role or roles assigned. To use the plugin with the **ansible-playbook** command, set the value of the **-i** option to the inventory file that uses the **freeipa** plugin. As a result, the plays in the playbook are executed only against those servers that have the roles specified in the inventory file.

The example below describes how to use the **freeipa** plugin to determine the versions of the **bind** package on the IdM servers in the topology that have the DNS role.

Procedure

1. Create a file, for example **inventory.yml**, to generate a dynamic inventory based on your selected IdM server role or roles. For example, to create an inventory of servers that have the IdM DNS server role:

```
---
plugin: freeipa
server: server.idm.example.com
ipaadmin_password: Secret123
verify: ca.crt
role: DNS server
```

- **server** defines the fully-qualified domain name of the host to start the scan.
 - **verify** defines the server TLS certificate file for verification.
 - **role** defines all the server roles that a host must have for the host to be returned.
2. Create a playbook, for example **bind-playbook.yml**, that you want to execute against the IdM servers identified by the **freeipa** inventory plugin. For example, to determine the release version of the **bind** package installed on the IdM DNS servers:


```

---
- name: Check bind version
  hosts: ipaservers

  tasks:
    - name: Query bind package version
      package:
        name: bind
        state: present
        register: bind_version

    - name: Display bind release info
      debug:
        msg: "Bind release: {{ bind_version.results[0].version }}"

```

- Optional: Create a graph representation of the IdM servers identified by the plugin:

```

$ ansible-inventory -i inventory.yml --graph
@all:
  |--@ungrouped
  |--@ipaservers:
  |   |--replica01.idm.example.com

```

- Run the Ansible playbook. Specify the inventory.yml file as inventory:

```

$ ansible-playbook -i inventory.yml bind-playbook.yml
[...]
TASK [Display bind release info]
ok: [10.0.193.174] => {
  "msg": "Bind release: 9.20.3"
}
[...]

```

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).

- [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](#)
- [Configuring a NETBIOS name for an IdM domain by using Ansible](#)
- [Ensuring that IdM users and groups have SIDs by using Ansible](#)

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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

Procedure

1. 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

  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
    - name: Query IPA global configuration
      ipaconfig:
        ipaadmin_password: "{{ ipaadmin_password }}"
```

```
register: serverconfig
```

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

2. Adapt the file by changing the following:

- The password of IdM administrator.
- Other values, if necessary.

3. Save the file.

4. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -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$guest_u:s0$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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the `secret.yml` Ansible vault stores your `ipaadmin_password`.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

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
  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml

  tasks:
    - name: set ca_renewal_master_server
      ipaconfig:
        ipaadmin_password: "{{ ipaadmin_password }}"
        ca_renewal_master_server: carenewal.idm.example.com

```

4. Adapt the file by changing:

- The password of IdM administrator set by the **ipaadmin_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, the file storing the password protecting the **secret.yml** file, and the inventory file:

```

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

```

Verification

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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

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
  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml

  tasks:
    # Set defaultlogin and maxusername
    - ipaconfig:
        ipaadmin_password: "{{ ipaadmin_password }}"
        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, the file storing the password protecting the **secret.yml** file, and the inventory file:

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

Verification

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.

6.4. CONFIGURING A NETBIOS NAME FOR AN IDM DOMAIN BY USING ANSIBLE

The NetBIOS name is used for Microsoft Windows' (SMB) type of sharing and messaging. You can use NetBIOS names to map a drive or connect to a printer.

Follow this procedure to use an Ansible playbook to configure a NetBIOS name for your Identity Management (IdM) domain.

Prerequisites

- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - The [ansible-freeipa](#) package is installed.

Assumptions

- The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
- The example assumes that the `secret.yml` Ansible vault stores your `ipaadmin_password` and that you know the vault file password.

Procedure

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

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

2. Create a `netbios-domain-name-present.yml` Ansible playbook file.

3. Add the following content to the file:

```
---
- name: Playbook to change IdM domain netbios name
  hosts: ipaserver
  become: no
  gather_facts: no

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml

  tasks:
  - name: Set IdM domain netbios name
    ipaconfig:
      ipaadmin_password: "{{ ipaadmin_password }}"
      netbios_name: IPADOM
```

4. Save the file.
5. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory netbios-  
domain-name-present.yml
```

When prompted, provide the vault file password.

Additional resources

- [Guidelines for configuring NetBIOS names](#)

6.5. ENSURING THAT IDM USERS AND GROUPS HAVE SIDS BY USING ANSIBLE

The Identity Management (IdM) server can assign unique security identifiers (SIDs) to IdM users and groups internally, based on the data from the ID ranges of the local domain. The SIDs are stored in the user and group objects.

The goal of ensuring that IdM users and groups have SIDs is to allow the generation of the Privileged Attribute Certificate (PAC), which is the first step towards IdM-IdM trusts. If IdM users and groups have SIDs, IdM is able to issue Kerberos tickets with PAC data.

Follow this procedure to achieve the following goals:

- Generate SIDs for already existing IdM users and user groups.
- Enable the generation of SIDs for IdM new users and groups.

Prerequisites

- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.

- The **ansible-freeipa** package is installed.

Assumptions

- The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
- The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password** and that you know the vault file password.

Procedure

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

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

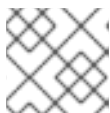
2. Create a `sids-for-users-and-groups-present.yml` Ansible playbook file.
3. Add the following content to the file:

```
---
- name: Playbook to ensure SIDs are enabled and users and groups have SIDs
  hosts: ipaserver
  become: no
  gather_facts: no

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml

  tasks:
  - name: Enable SID and generate users and groups SIDS
    ipaconfig:
      ipaadmin_password: "{{ ipaadmin_password }}"
      enable_sid: true
      add_sids: true
```

The **enable_sid** variable enables SID generation for future IdM users and groups. The **add_sids** variable generates SIDs for existing IdM users and groups.



NOTE

When using **add_sids: true**, you must also set the **enable_sid** variable to **true**.

4. Save the file.
5. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory sids-for-users-and-groups-present.yml
```

When prompted, provide the vault file password.

Additional resources

- [The role of security and relative identifiers in IdM ID ranges](#) .

6.6. ADDITIONAL RESOURCES

- See **README-config.md** in the `/usr/share/doc/ansible-freeipa/` directory.
- See sample playbooks in the `/usr/share/doc/ansible-freeipa/playbooks/config` directory.

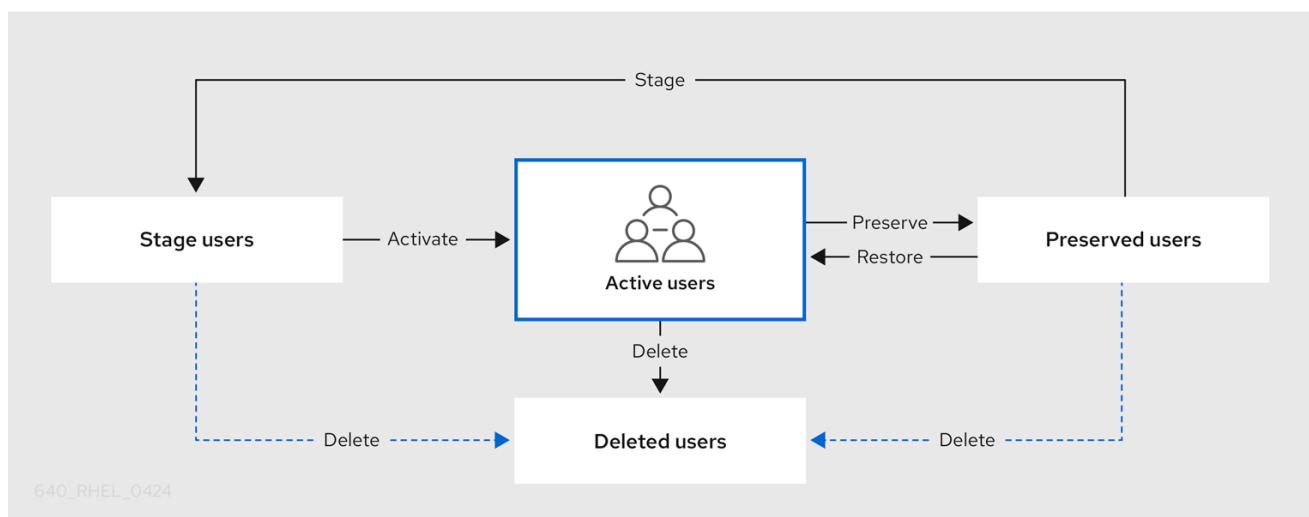
CHAPTER 7. MANAGING USER ACCOUNTS USING ANSIBLE PLAYBOOKS

You can manage users in IdM using Ansible playbooks. After presenting the [user life cycle](#), learn how to use Ansible playbooks to ensure the presence or absence of users listed directly in the **YML** file.

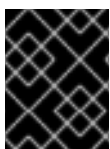
7.1. USER LIFE CYCLE

Identity Management (IdM) 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Create user idm_user
    ipauser:
      ipaadmin_password: "{{ ipaadmin_password }}"
      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 --vault-password-file=password_file -v -i
path_to_inventory_directory/inventory.file path_to_playbooks_directory/add-IdM-
user.yml

```

Verification

- 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the *~/MyPlaybooks/* directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

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

  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
```

```

- name: Create user idm_users
  ipauser:
    ipaadmin_password: "{{ ipaadmin_password }}"
    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 resets 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 resets password.

3. Run the playbook:

```

$ ansible-playbook --vault-password-file=password_file -v -i
  path_to_inventory_directory/inventory.file path_to_playbooks_directory/add-
  users.yml

```

Verification

- 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the *~/MyPlaybooks/* directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

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/README-user.md** file:

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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Include users_present.json
    include_vars:
      file: users_present.json

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

1. 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/README-user.md** 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": "First 1",
      "last": "Last 1",
      "password": "Password123"
    },
    {
      "name": "idm_user_2",
      "first": "First 2",
      "last": "Last 2"
    },
    {
      "name": "idm_user_3",
      "first": "First 3",
      "last": "Last 3"
    }
  ]
}
```

2. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

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

Verification

- You can verify if the user accounts are present in IdM 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.

7.5. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the `secret.yml` Ansible vault stores your `ipaadmin_password`.
- The target node, that is the node on which the `ansible-freeipa` module is executed, is part of the IdM domain as an IdM client, server or replica.

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 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Delete users idm_user_1, idm_user_2, idm_user_3
    ipauser:
      ipaadmin_password: "{{ ipaadmin_password }}"
      users:
        - name: idm_user_1
        - name: idm_user_2
        - name: idm_user_3
      state: absent
```

3. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the `secret.yml` file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i
path_to_inventory_directory/inventory.file path_to_playbooks_directory/delete-
users.yml
```

Verification

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.

7.6. ADDITIONAL RESOURCES

- See the **README-user.md** Markdown file in the `/usr/share/doc/ansible-freeipa/` directory.
- 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

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

Group name	Default group members
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](#).

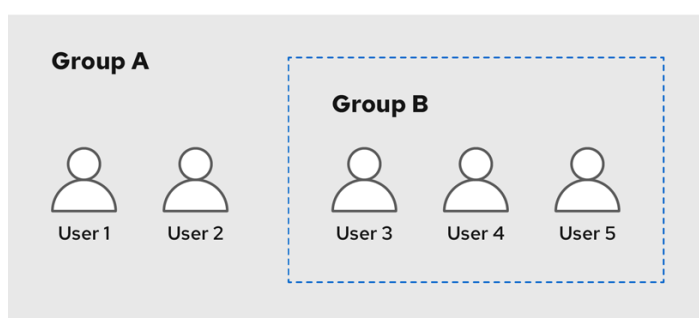
8.2. DIRECT AND INDIRECT GROUP MEMBERS

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



640_RHEL_0424

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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the `secret.yml` Ansible vault stores your `ipaadmin_password`.
- The target node, that is the node on which the `ansible-freeipa` module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Create group ops with gid 1234
    ipagroup:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: ops
      gidnumber: 1234

  - name: Create group sysops
    ipagroup:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: sysops
      user:
      - idm_user

  - name: Create group appops
    ipagroup:
      ipaadmin_password: "{{ ipaadmin_password }}"
```

```

    name: appops

- name: Add group members sysops and appops to group ops
  ipagroup:
    ipaadmin_password: "{{ ipaadmin_password }}"
    name: ops
    group:
      - sysops
      - appops

```

3. Run the playbook:

```

$ ansible-playbook --vault-password-file=password_file -v -i
  path_to_inventory_directory/inventory.file path_to_playbooks_directory/add-group-
  members.yml

```

Verification

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

- See the [/usr/share/doc/ansible-freeipa/README-group.md](#) Markdown file.

8.4. USING ANSIBLE TO ADD MULTIPLE IDM GROUPS IN A SINGLE TASK

You can use the **ansible-freeipa ipagroup** module to add, modify, and delete multiple Identity Management (IdM) user groups with a single Ansible task. For that, use the **groups** option of the **ipagroup** module.

Using the **groups** option, you can also specify multiple group variables that only apply to a particular group. Define this group by the **name** variable, which is the only mandatory variable for the **groups** option.

Complete this procedure to ensure the presence of the **sysops** and the **appops** groups in IdM in a single task. Define the **sysops** group as a nonposix group and the **appops** group as an external group.

Prerequisites

- On the control node:
 - You are using Ansible version 2.14 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 in the `~/MyPlaybooks/` directory.
 - You are using RHEL 9.3 and later.
 - You have stored your `ipaadmin_password` in the `secret.yml` Ansible vault.

Procedure

1. Create your Ansible playbook file `add-nonposix-and-external-groups.yml` with the following content:

```
---
- name: Playbook to add nonposix and external groups
  hosts: ipaserver
  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml

  tasks:
    - name: Add nonposix group sysops and external group appops
      ipagroup:
        ipaadmin_password: "{{ ipaadmin_password }}"
        groups:
          - name: sysops
            nonposix: true
          - name: appops
            external: true
```

2. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i
<path_to_inventory_directory>/hosts <path_to_playbooks_directory>/add-nonposix-
and-external-groups.yml
```

Additional resources

- [The group module in ansible-freeipa upstream docs](#)

8.5. USING ANSIBLE TO ENABLE AD USERS TO ADMINISTER IDM

Follow this procedure to use an Ansible playbook to ensure that a user ID override is present in an Identity Management (IdM) group. The user ID override is the override of an Active Directory (AD) user that you created in the Default Trust View after you established a trust with AD. As a result of running the playbook, an AD user, for example an AD administrator, is able to fully administer IdM without having two different accounts and passwords.

Prerequisites

- You know the IdM **admin** password.
- You have [installed a trust with AD](#).
- The user ID override of the AD user already exists in IdM. If it does not, create it with the **ipa idoverrideuser-add 'default trust view' ad_user@ad.example.com** command.
- The [group to which you are adding the user ID override already exists in IdM](#).
- You are using the 4.8.7 version of IdM or later. To view the version of IdM you have installed on your server, enter **ipa --version**.
- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

Procedure

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

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

2. Create an **add-useridoverride-to-group.yml** playbook with the following content:

```
---
- name: Playbook to ensure presence of users in a group
  hosts: ipaserver

  - name: Ensure the ad_user@ad.example.com user ID override is a member of the admins
    group:
      ipagroup:
        ipaadmin_password: "{{ ipaadmin_password }}"
        name: admins
        idoverrideuser:
          - ad_user@ad.example.com
```

In the example:

- Secret123 is the IdM **admin** password.
- **admins** is the name of the IdM POSIX group to which you are adding the **ad_user@ad.example.com** ID override. Members of this group have full administrator privileges.

- **ad_user@ad.example.com** is the user ID override of an AD administrator. The user is stored in the AD domain with which a trust has been established.
3. Save the file.
 4. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory add-  
useridoverride-to-group.yml
```

Additional resources

- [ID overrides for AD users](#)
- [/usr/share/doc/ansible-freeipa/README-group.md](#)
- [/usr/share/doc/ansible-freeipa/playbooks/user](#)
- [Using ID views in Active Directory environments](#)
- [Enabling AD users to administer IdM](#)

8.6. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the **~/MyPlaybooks/** directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure user test is present for group_a
    ipagroup:
      ipadmin_password: "{{ ipadmin_password }}"
      name: group_a
      membermanager_user: test

  - name: Ensure group_admins is present for group_a
    ipagroup:
      ipadmin_password: "{{ ipadmin_password }}"
      name: group_a
      membermanager_group: group_admins
```

3. Run the playbook:

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

Verification

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 on your system.

8.7. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the `secret.yml` Ansible vault stores your `ipaadmin_password`.
- The target node, that is the node on which the `ansible-freeipa` module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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

  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
    - name: Ensure member manager user and group members are absent for group_a
      ipagroup:
        ipaadmin_password: "{{ ipaadmin_password }}"
        name: group_a
        membermanager_user: test
        membermanager_group: group_admins
        action: member
        state: absent
```

3. Run the playbook:

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

Verification

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 on your system.

CHAPTER 9. USING ANSIBLE TO AUTOMATE GROUP MEMBERSHIP IN IDM

Using automatic group membership, you can assign users and hosts user groups and host groups automatically, based on their attributes. For example, you can:

- Divide employees' user entries into groups based on the employees' manager, location, position or any other attribute. You can list all attributes by entering **ipa user-add --help** on the command-line.
- Divide hosts into groups based on their class, location, or any other attribute. You can list all attributes by entering **ipa host-add --help** on the command-line.
- Add all users or all hosts to a single global group.

You can use Red Hat Ansible Engine to automate the management of automatic group membership in Identity Management (IdM).

9.1. USING ANSIBLE TO ENSURE THAT AN AUTOMEMBER RULE FOR AN IDM USER GROUP IS PRESENT

The following procedure describes how to use an Ansible playbook to ensure an **automember** rule for an Identity Management (IdM) group exists. In the example, the presence of an **automember** rule is ensured for the **testing_group** user group.

Prerequisites

- You know the IdM **admin** password.
- The **testing_group** user group exists in IdM.
- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the **~/MyPlaybooks/** directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

Procedure

1. Navigate to your **~/MyPlaybooks/** directory:

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

2. Copy the **automember-group-present.yml** Ansible playbook file located in the **/usr/share/doc/ansible-freeipa/playbooks/automember/** directory:

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

3. Open the **automember-group-present-copy.yml** file for editing.
4. Adapt the file by setting the following variables in the **ipaaautomember** task section:
 - Set the **ipaadmin_password** variable to the password of the IdM **admin**.
 - Set the **name** variable to **testing_group**.
 - Set the **automember_type** variable to **group**.
 - Ensure that the **state** variable is set to **present**.

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

```
---
- name: Automember group present example
  hosts: ipaserver
  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
    - name: Ensure group automember rule admins is present
      ipaaautomember:
        ipaadmin_password: "{{ ipaadmin_password }}"
        name: testing_group
        automember_type: group
        state: present
```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory automember-group-present-copy.yml
```

9.2. USING ANSIBLE TO ENSURE THAT A SPECIFIED CONDITION IS PRESENT IN AN IDM USER GROUP AUTOMEMBER RULE

Additional resources

The following procedure describes how to use an Ansible playbook to ensure that a specified condition exists in an **automember** rule for an Identity Management (IdM) group. In the example, the presence of a UID-related condition in the **automember** rule is ensured for the **testing_group** group. By specifying the **.*** condition, you ensure that all future IdM users automatically become members of the **testing_group**.

Prerequisites

- You know the IdM **admin** password.
- The **testing_group** user group and automember user group rule exist in IdM.

- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

Procedure

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

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

2. Copy the **automember-hostgroup-rule-present.yml** Ansible playbook file located in the `/usr/share/doc/ansible-freeipa/playbooks/automember/` directory and name it, for example, **automember-usergroup-rule-present.yml**:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/automember/automember-hostgroup-rule-present.yml automember-usergroup-rule-present.yml
```

3. Open the **automember-usergroup-rule-present.yml** file for editing.
4. Adapt the file by modifying the following parameters:
 - Rename the playbook to correspond to your use case, for example: **Automember user group rule member present**.
 - Rename the task to correspond to your use case, for example: **Ensure an automember condition for a user group is present**.
 - Set the following variables in the **ipaautomember** task section:
 - Set the **ipaadmin_password** variable to the password of the IdM **admin**.
 - Set the **name** variable to **testing_group**.
 - Set the **automember_type** variable to **group**.
 - Ensure that the **state** variable is set to **present**.
 - Ensure that the **action** variable is set to **member**.
 - Set the **inclusive key** variable to **UID**.
 - Set the **inclusive expression** variable to **.***

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

```
---
```



```

- name: Automember user group rule member present
  hosts: ipaserver
  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure an automember condition for a user group is present
    ipaautomember:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: testing_group
      automember_type: group
      state: present
      action: member
      inclusive:
        - key: UID
          expression: .*

```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory automember-usergroup-rule-present.yml
```

Verification

1. Log in as an IdM administrator.

```
$ kinit admin
```

2. Add a user, for example:

```

$ ipa user-add user101 --first user --last 101
-----
Added user "user101"
-----
User login: user101
First name: user
Last name: 101
...
Member of groups: ipausers, testing_group
...

```

9.3. USING ANSIBLE TO ENSURE THAT A CONDITION IS ABSENT FROM AN IDM USER GROUP AUTOMEMBER RULE

Additional resources

The following procedure describes how to use an Ansible playbook to ensure a condition is absent from an **automember** rule for an Identity Management (IdM) group. In the example, the absence of a condition in the **automember** rule is ensured that specifies that users whose **initials** are **dp** should be included. The automember rule is applied to the **testing_group** group. By applying the condition, you ensure that no future IdM user whose initials are **dp** becomes a member of the **testing_group**.

Prerequisites

- You know the IdM **admin** password.
- The **testing_group** user group and automember user group rule exist in IdM.
- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

Procedure

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

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

2. Copy the **automember-hostgroup-rule-absent.yml** Ansible playbook file located in the `/usr/share/doc/ansible-freeipa/playbooks/automember/` directory and name it, for example, **automember-usergroup-rule-absent.yml**:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/automember/automember-hostgroup-rule-absent.yml automember-usergroup-rule-absent.yml
```

3. Open the **automember-usergroup-rule-absent.yml** file for editing.
4. Adapt the file by modifying the following parameters:
 - Rename the playbook to correspond to your use case, for example: **Automember user group rule member absent**.
 - Rename the task to correspond to your use case, for example: **Ensure an automember condition for a user group is absent**.
 - Set the following variables in the **ipaautomember** task section:
 - Set the **ipaadmin_password** variable to the password of the IdM **admin**.
 - Set the **name** variable to **testing_group**.
 - Set the **automember_type** variable to **group**.
 - Ensure that the **state** variable is set to **absent**.
 - Ensure that the **action** variable is set to **member**.
 - Set the **inclusive key** variable to **initials**.

- Set the **inclusive expression** variable to **dp**.

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

```
---
- name: Automember user group rule member absent
  hosts: ipaserver
  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
    - name: Ensure an automember condition for a user group is absent
      ipaautomember:
        ipaadmin_password: "{{ ipaadmin_password }}"
        name: testing_group
        automember_type: group
        state: absent
        action: member
        inclusive:
          - key: initials
            expression: dp
```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory automember-
usergroup-rule-absent.yml
```

Verification

1. Log in as an IdM administrator.

```
$ kinit admin
```

2. View the automember group:

```
$ ipa automember-show --type=group testing_group
Automember Rule: testing_group
```

The absence of an **Inclusive Regex: initials=dp** entry in the output confirms that the **testing_group** automember rule does not contain the condition specified.

9.4. USING ANSIBLE TO ENSURE THAT AN AUTOMEMBER RULE FOR AN IDM USER GROUP IS ABSENT

Additional resources

The following procedure describes how to use an Ansible playbook to ensure an **automember** rule is absent for an Identity Management (IdM) group. In the example, the absence of an **automember** rule is ensured for the **testing_group** group.

Deleting an automember rule also deletes all conditions associated with the rule. To remove only specific conditions from a rule, see [Using Ansible to ensure that a condition is absent in an IdM user group automember rule](#).

Prerequisites

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the `secret.yml` Ansible vault stores your `ipaadmin_password`.
- The target node, that is the node on which the `ansible-freeipa` module is executed, is part of the IdM domain as an IdM client, server or replica.

Procedure

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

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

2. Copy the `automember-group-absent.yml` Ansible playbook file located in the `/usr/share/doc/ansible-freeipa/playbooks/automember/` directory:

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

3. Open the `automember-group-absent-copy.yml` file for editing.
4. Adapt the file by setting the following variables in the `ipaautomember` task section:
 - Set the `ipaadmin_password` variable to the password of the IdM `admin`.
 - Set the `name` variable to `testing_group`.
 - Set the `automember_type` variable to `group`.
 - Ensure that the `state` variable is set to `absent`.

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

```
---
- name: Automember group absent example
  hosts: ipaserver
  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
    - name: Ensure group automember rule admins is absent
      ipaautomember:
        ipaadmin_password: "{{ ipaadmin_password }}"
```

```
name: testing_group
automember_type: group
state: absent
```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory automember-
group-absent.yml
```

Additional resources

- See the **README-automember.md** file in the `/usr/share/doc/ansible-freeipa/` directory.
- See the `/usr/share/doc/ansible-freeipa/playbooks/automember` directory.

9.5. USING ANSIBLE TO ENSURE THAT A CONDITION IS PRESENT IN AN IDM HOST GROUP AUTOMEMBER RULE

Follow this procedure to use Ansible to ensure that a condition is present in an IdM host group automember rule. The example describes how to ensure that hosts with the **FQDN** of `*.idm.example.com` are members of the `primary_dns_domain_hosts` host group and hosts whose **FQDN** is `*.example.org` are not members of the `primary_dns_domain_hosts` host group.

Prerequisites

- You know the IdM **admin** password.
- The `primary_dns_domain_hosts` host group and automember host group rule exist in IdM.
- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

Procedure

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

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

2. Copy the **automember-hostgroup-rule-present.yml** Ansible playbook file located in the `/usr/share/doc/ansible-freeipa/playbooks/automember/` directory:

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

3. Open the **automember-hostgroup-rule-present-copy.yml** file for editing.
4. Adapt the file by setting the following variables in the **ipaaautomember** task section:
 - Set the **ipaadmin_password** variable to the password of the IdM **admin**.
 - Set the **name** variable to **primary_dns_domain_hosts**.
 - Set the **automember_type** variable to **hostgroup**.
 - Ensure that the **state** variable is set to **present**.
 - Ensure that the **action** variable is set to **member**.
 - Ensure that the **inclusive key** variable is set to **fqdn**.
 - Set the corresponding **inclusive expression** variable to ***.idm.example.com**.
 - Set the **exclusive key** variable to **fqdn**.
 - Set the corresponding **exclusive expression** variable to ***.example.org**.

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

```
---
- name: Automember user group rule member present
  hosts: ipaserver
  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
    - name: Ensure an automember condition for a user group is present
      ipaaautomember:
        ipaadmin_password: "{{ ipaadmin_password }}"
        name: primary_dns_domain_hosts
        automember_type: hostgroup
        state: present
        action: member
        inclusive:
          - key: fqdn
            expression: *.idm.example.com
        exclusive:
          - key: fqdn
            expression: *.example.org
```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory automember-hostgroup-rule-present-copy.yml
```

Additional resources

- See the **README-automember.md** file in the `/usr/share/doc/ansible-freeipa/` directory.
- See the `/usr/share/doc/ansible-freeipa/playbooks/automember` directory.

CHAPTER 10. USING ANSIBLE PLAYBOOKS TO MANAGE SELF-SERVICE RULES IN IDM

Learn about self-service rules in Identity Management (IdM) and how to create and edit self-service access rules using Ansible playbooks. With self-service access control rules, an IdM entity can perform specified operations on its IdM Directory Server entry.

10.1. 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.

10.2. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the **~/MyPlaybooks/** directory, you have created an **Ansible inventory file** with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure self-service rule "Users can manage their own name details" is present
    ipaselfservice:
      ipaadmin_password: "{{ ipaadmin_password }}"
      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. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory selfservice-
present-copy.yml
```

Additional resources

- [Self-service access control in IdM](#)
- The **README-selfservice.md** file in the **/usr/share/doc/ansible-freeipa/** directory

- The `/usr/share/doc/ansible-freeipa/playbooks/selfservice` directory

10.3. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the `secret.yml` Ansible vault stores your `ipaadmin_password`.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

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

  vars_files:
```

```
- /home/user_name/MyPlaybooks/secret.yml
tasks:
- name: Ensure self-service rule "Users can manage their own name details" is absent
  ipaselfservice:
    ipaadmin_password: "{{ ipaadmin_password }}"
    name: "Users can manage their own name details"
    state: absent
```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory selfservice-
absent-copy.yml
```

Additional resources

- [Self-service access control in IdM](#)
- The **README-selfservice.md** file in the `/usr/share/doc/ansible-freeipa/` directory
- Sample playbooks in the `/usr/share/doc/ansible-freeipa/playbooks/selfservice` directory

10.4. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure selfservice "Users can manage their own name details" member attribute
    surname is present
    ipaselfservice:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: "Users can manage their own name details"
      attribute:
      - surname
      action: member
```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory selfservice-member-present-copy.yml
```

Additional resources

- [Self-service access control in IdM](#)
- The **README-selfservice.md** file available in the **/usr/share/doc/ansible-freeipa/** directory
- The sample playbooks in the **/usr/share/doc/ansible-freeipa/playbooks/selfservice** directory

10.5. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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 **ipadmin_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

  vars_files:
```

```
- /home/user_name/MyPlaybooks/secret.yml
tasks:
- name: Ensure selfservice "Users can manage their own name details" member attributes
  givenname and surname are absent
  ipaselfservice:
    ipaadmin_password: "{{ ipaadmin_password }}"
    name: "Users can manage their own name details"
    attribute:
      - givenname
      - surname
    action: member
    state: absent
```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory selfservice-
member-absent-copy.yml
```

Additional resources

- [Self-service access control in IdM](#)
- The **README-selfservice.md** file in the **/usr/share/doc/ansible-freeipa/** directory
- The sample playbooks in the **/usr/share/doc/ansible-freeipa/playbooks/selfservice** directory

CHAPTER 11. 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.

11.1. 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.

11.2. 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.

11.3. 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**
- **employeeetype**

Prerequisites

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the `secret.yml` Ansible vault stores your `ipaadmin_password`.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure delegation "basic manager attributes" is present
    ipadelegation:
      ipaadmin_password: "{{ ipaadmin_password }}"
      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. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i ~/MyPlaybooks/inventory
delegation-present-copy.yml
```

Additional resources

- [Delegation rules](#)
- The **README-delegation.md** file in the **/usr/share/doc/ansible-freeipa/** directory

- Sample playbooks in the `/usr/share/doc/ansible-freeipa/playbooks/ipadelegation` directory

11.4. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

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

  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
```

```

tasks:
- name: Ensure delegation "basic manager attributes" is absent
  ipadelegation:
    ipaadmin_password: "{{ ipaadmin_password }}"
    name: "basic manager attributes"
    state: absent

```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```

$ ansible-playbook --vault-password-file=password_file -v -i ~/MyPlaybooks/inventory
delegation-absent-copy.yml

```

Additional resources

- [Delegation rules](#)
- The **README-delegation.md** file in the `/usr/share/doc/ansible-freeipa/` directory
- Sample playbooks in the `/usr/share/doc/ansible-freeipa/playbooks/ipadelegation` directory

11.5. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure delegation "basic manager attributes" member attribute departmentnumber
    is present
    ipadelegation:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: "basic manager attributes"
      attribute:
      - departmentnumber
      action: member
```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i ~/MyPlaybooks/inventory delegation-member-present-copy.yml
```

Additional resources

- [Delegation rules](#)
- The **README-delegation.md** file in the **/usr/share/doc/ansible-freeipa/** directory
- The sample playbooks in the **/usr/share/doc/ansible-freeipa/playbooks/ipadelegation** directory

11.6. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure delegation "basic manager attributes" member attributes employeeenumber
    and employeetype are absent
    ipadelegation:
      ipadmin_password: "{{ ipadmin_password }}"
      name: "basic manager attributes"
      attribute:
        - employeeenumber
        - employeetype
      action: member
      state: absent
```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i ~/MyPlaybooks/inventory
delegation-member-absent-copy.yml
```

Additional resources

- [Delegation rules](#)
- The **README-delegation.md** file in the `/usr/share/doc/ansible-freeipa/` directory.
- Sample playbooks in the `/usr/share/doc/ansible-freeipa/playbooks/ipadelegation` directory

CHAPTER 12. 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, and enabling read-access.
- **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.

Learn about operations you can perform when managing RBAC using Ansible playbooks.

12.1. 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**

**NOTE**

Setting the **memberof** attribute permission is not applied if the target LDAP entry does not contain any reference to group membership.

- **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.

12.2. 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.

12.3. 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.

12.4. ROLES IN IDM

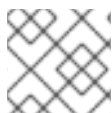
A role is a list of privileges that users specified for the role possess.

In effect, permissions grant the ability to perform given low-level tasks (such as creating a user entry and adding an entry to a group), 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.

12.5. PREDEFINED ROLES IN IDENTITY MANAGEMENT

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

Table 12.1. Predefined Roles in Identity Management

Role	Privilege	Description
Enrollment Administrator	Host Enrollment	Responsible for client, or host, enrollment
helpdesk	Modify Users and Reset passwords, Modify Group membership	Responsible for performing simple user administration tasks
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

Role	Privilege	Description
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

12.6. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

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: true
  gather_facts: no

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - iparole:
      ipaadmin_password: "{{ ipaadmin_password }}"
      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. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

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

Additional resources

- [Encrypting content with Ansible Vault](#)
- [Roles in IdM](#)

- The **README-role** file in the `/usr/share/doc/ansible-freeipa/` directory
- Sample playbooks in the `/usr/share/doc/ansible-freeipa/playbooks/iparole` directory

12.7. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

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: true
  gather_facts: no

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - iparole:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: user_and_host_administrator
      state: absent

```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```

$ ansible-playbook --vault-password-file=password_file -v -i
~/<MyPlaybooks>/inventory role-is-absent-copy.yml

```

Additional resources

- [Encrypting content with Ansible Vault](#)
- [Roles in IdM](#)
- The **README-role** Markdown file in the `/usr/share/doc/ansible-freeipa/` directory
- Sample playbooks in the `/usr/share/doc/ansible-freeipa/playbooks/iparole` directory

12.8. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

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: true
  gather_facts: no

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - iparole:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: helpdesk
      group: junior_sysadmins
      action: member
```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

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

Additional resources

- [Encrypting content with Ansible Vault](#)
- [Roles in IdM](#)

- The **README-role** Markdown file in the `/usr/share/doc/ansible-freeipa/` directory
- Sample playbooks in the `/usr/share/doc/ansible-freeipa/playbooks/iparole` directory

12.9. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the `secret.yml` Ansible vault stores your `ipadmin_password`.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

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 `ipadmin_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: true
  gather_facts: no

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - iparole:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: helpdesk
      user
      - user_01
      - user_02
      action: member
      state: absent
```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

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

Additional resources

- [Encrypting content with Ansible Vault](#)
- [Roles in IdM](#)
- The **README-role** Markdown file in the `/usr/share/doc/ansible-freeipa/` directory
- Sample playbooks in the `/usr/share/doc/ansible-freeipa/playbooks/iparole` directory

12.10. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.

- The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
- The example assumes that the `secret.yml` Ansible vault stores your `ipaadmin_password`.
- The target node, that is the node on which the `ansible-freeipa` module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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: true
  gather_facts: no

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - iparole:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: web_administrator
      service:
      - HTTP/client01.idm.example.com
      action: member
```

5. Save the file.

- Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

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

Additional resources

- [Encrypting content with Ansible Vault](#)
- [Roles in IdM](#)
- The **README-role** Markdown file in the `/usr/share/doc/ansible-freeipa/` directory
- The sample playbooks in the `/usr/share/doc/ansible-freeipa/playbooks/iparole` directory

12.11. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- The **web_administrator** role exists in IdM.
- The **client01.idm.example.com** host exists in IdM.

Procedure

- Navigate to the `~/<MyPlaybooks>/` directory:

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

- 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: true
  gather_facts: no

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - iparole:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: web_administrator
      host:
      - client01.idm.example.com
      action: member
```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

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

Additional resources

- [Encrypting content with Ansible Vault](#)
- [Roles in IdM](#)
- The **README-role** Markdown file in the **/usr/share/doc/ansible-freeipa/** directory
- Sample playbooks in the **/usr/share/doc/ansible-freeipa/playbooks/iparole** directory

12.12. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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: true
  gather_facts: no

  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
```

```
- iparole:
  ipaadmin_password: "{{ ipaadmin_password }}"
  name: web_administrator
  hostgroup:
  - web_servers
  action: member
```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

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

Additional resources

- [Encrypting content with Ansible Vault](#)
- [Roles in IdM](#)
- The **README-role** Markdown file in the `/usr/share/doc/ansible-freeipa/` directory
- The sample playbooks in the `/usr/share/doc/ansible-freeipa/playbooks/iparole` directory

CHAPTER 13. 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.

Learn how to use Ansible playbooks to manage RBAC privileges in Identity Management (IdM).

Prerequisites

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

13.1. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure privilege full_host_administration is present
    ipaprivilege:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: full_host_administration
      description: This privilege combines all IdM permissions related to host
        administration
```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory privilege-
present-copy.yml
```

13.2. 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

- On the control node:
 - You are using Ansible version 2.14 or later.

- You have installed the **ansible-freeipa** package.
- The example assumes that in the `~/MyPlaybooks/` directory, you have created an **Ansible inventory file** with the fully-qualified domain name (FQDN) of the IdM server.
- The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- The **full_host_administration** privilege exists. For information about 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

  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
    - name: Ensure that permissions are present for the "full_host_administration" privilege
      ipaprivilege:
        ipaadmin_password: "{{ ipaadmin_password }}"
        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. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory privilege-member-present-copy.yml
```

13.3. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.

- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

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

  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
    - name: Ensure that the "Request Certificate ignoring CA ACLs" permission is absent from
      the "Certificate Administrators" privilege
      ipaprivilege:
        ipaadmin_password: "{{ ipaadmin_password }}"
        name: Certificate Administrators
        permission:
          - "Request Certificate ignoring CA ACLs"
        action: member
        state: absent
```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory privilege-  
member-absent-copy.yml
```

13.4. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- The **full_host_administration** privilege exists. For more information about 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 **ipadmin_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
```

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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure the full_host_administration privilege is renamed to
    limited_host_administration
    ipaprivilege:
      ipadmin_password: "{{ ipadmin_password }}"
      name: full_host_administration
      rename: limited_host_administration
      state: renamed
```

7. Save the file.
8. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory rename-
privilege.yml
```

13.5. 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

- On the control node:
 - You are using Ansible version 2.14 or later.

- You have installed the **ansible-freeipa** package.
- The example assumes that in the `~/MyPlaybooks/` directory, you have created an **Ansible inventory file** with the fully-qualified domain name (FQDN) of the IdM server.
- The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure privilege "CA administrator" is absent
    ipaprivilege:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: CA administrator
      state: absent
```

6. Save the file.
7. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory privilege-absent-copy.yml
```

13.6. ADDITIONAL RESOURCES

- See [Privileges in IdM](#).
- See [Permissions in IdM](#).
- See the **README-privilege** file available in the `/usr/share/doc/ansible-freeipa/` directory.
- See the sample playbooks in the `/usr/share/doc/ansible-freeipa/playbooks/ipaprivilege` directory.

CHAPTER 14. 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.

Learn about operations you can perform when managing RBAC permissions in Identity Management (IdM) using Ansible playbooks:

Prerequisites

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

14.1. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.

- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure that the "MyPermission" permission is present
    ipapermission:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: MyPermission
      object_type: host
      right: all
```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory permission-
present-copy.yml
```

14.2. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

Procedure

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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure that the "MyPermission" permission is present with an attribute
    ipapermission:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: MyPermission
      object_type: host
      right: all
      attrs: description
```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the `secret.yml` file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory permission-
present-with-attribute.yml
```

Additional resources

- See [User and group schema](#) in *Linux Domain Identity, Authentication and Policy Guide* in RHEL 7.

14.3. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the `secret.yml` Ansible vault stores your `ipadmin_password`.
- The target node, that is the node on which the `ansible-freeipa` module is executed, is part of the IdM domain as an IdM client, server or replica.

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 `ipadmin_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
```

```
vars_files:
- /home/user_name/MyPlaybooks/secret.yml
tasks:
- name: Ensure that the "MyPermission" permission is absent
  ipapermission:
    ipaadmin_password: "{{ ipaadmin_password }}"
    name: MyPermission
    state: absent
```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory permission-absent-copy.yml
```

14.4. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the **~/MyPlaybooks/** directory, you have created an **Ansible inventory file** with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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

  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
    - name: Ensure that the "gecos" and "description" attributes are present in
      "MyPermission"
      ipapermission:
        ipaadmin_password: "{{ ipaadmin_password }}"
        name: MyPermission
        attrs:
          - description
          - gecoss
        action: member
```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the `secret.yml` file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory permission-member-present-copy.yml
```

14.5. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the `secret.yml` Ansible vault stores your `ipaadmin_password`.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure that an attribute is not a member of "MyPermission"
    ipapermission:
      ipadmin_password: "{{ ipadmin_password }}"
      name: MyPermission
      attrs: description
      action: member
      state: absent
```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory permission-
member-absent-copy.yml
```

14.6. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the **~/MyPlaybooks/** directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

- 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Rename the "MyPermission" permission
    ipapermission:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: MyPermission
      rename: MyNewPermission
      state: renamed
```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory permission-
renamed-copy.yml
```

14.7. ADDITIONAL RESOURCES

- See [Permissions in IdM](#).

- See [Privileges in IdM](#).
- See the **README-permission** file available in the `/usr/share/doc/ansible-freeipa/` directory.
- See the sample playbooks in the `/usr/share/doc/ansible-freeipa/playbooks/ipapermission` directory.

CHAPTER 15. 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 Ansible to manage IdM replication agreements, topology segments, and topology suffixes.

15.1. 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.

Follow this procedure 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 [Guidelines for connecting IdM replicas in a topology](#).
- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the **~/MyPlaybooks/** directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password** and that you have access to a file that stores the password protecting the **secret.yml** file.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

Procedure

1. Navigate to your **~/MyPlaybooks/** directory:

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

- Copy the **add-topologysegment.yml** Ansible playbook file provided by the **ansible-freeipa** package:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/topology/add-topologysegment.yml
add-topologysegment-copy.yml
```

- Open the **add-topologysegment-copy.yml** file for editing.
- Adapt the file by setting the following variables in the **ipatopologysegment** task section:
 - Indicate that the value of the **ipaadmin_password** variable is defined in the **secret.yml** Ansible vault file.
 - 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Add topology segment
    ipatopologysegment:
      ipaadmin_password: "{{ ipaadmin_password }}"
      suffix: domain
      left: server.idm.example.com
      right: replica.idm.example.com
      state: present
```

- Save the file.
- Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory add-
topologysegment-copy.yml
```

Additional resources

- [Explaining Replication Agreements, Topology Suffixes, and Topology Segments](#)
- /usr/share/doc/ansible-freeipa/README-topology.md**

- Sample playbooks in **/usr/share/doc/ansible-freeipa/playbooks/topology**

15.2. 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.

Follow this procedure 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 have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the **~/MyPlaybooks/** directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password** and that you have access to a file that stores the password protecting the **secret.yml** file.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

Procedure

1. Navigate to your **~/MyPlaybooks/** directory:

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

2. Copy the **add-topologysegments.yml** Ansible playbook file provided by the **ansible-freeipa** package:

```
$ 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:
 - Indicate that the value of the **ipaadmin_password** variable is defined in the **secret.yml** Ansible vault file.
 - 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
  gather_facts: false

  vars:
    ipaadmin_password: "{{ ipaadmin_password }}"
    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 }

  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  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
      loop: "{{ ipatopology_segments | default([]) }}"
```

6. Save the file.
7. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory add-topologysegments-copy.yml
```

Additional resources

- [Explaining Replication Agreements, Topology Suffixes, and Topology Segments](#)
- **/usr/share/doc/ansible-freeipa/README-topology.md**
- Sample playbooks in **/usr/share/doc/ansible-freeipa/playbooks/topology**

15.3. 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.

Follow this procedure to verify that replication agreements exist between multiple pairs of replicas in IdM. In contrast to [Using Ansible to ensure a replication agreement exists in IdM](#), this procedure does not modify the existing configuration.

Prerequisites

- Ensure that you understand the recommendations for designing your Identity Management (IdM) topology listed in [Connecting the replicas in a topology](#).
- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the `secret.yml` Ansible vault stores your `ipaadmin_password` and that you have access to a file that stores the password protecting the `secret.yml` file.
- The target node, that is the node on which the `ansible-freeipa` module is executed, is part of the IdM domain as an IdM client, server or replica.

Procedure

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

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

2. Copy the `check-topologysegments.yml` Ansible playbook file provided by the `ansible-freeipa` package:

```
$ 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:

- Indicate that the value of the `ipaadmin_password` variable is defined in the `secret.yml` Ansible vault file.
- 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
  gather_facts: false

  vars:
    ipaadmin_password: "{{ ipaadmin_password }}"
    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 }

  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  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. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory check-topologysegments-copy.yml
```

Additional resources

- [Explaining Replication Agreements, Topology Suffixes, and Topology Segments](#)
- **/usr/share/doc/ansible-freeipa/README-topology.md**
- Sample playbooks in **/usr/share/doc/ansible-freeipa/playbooks/topology**

15.4. 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 data about 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.

Follow this procedure 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 have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the `secret.yml` Ansible vault stores your `ipaadmin_password` and that you have access to a file that stores the password protecting the `secret.yml` file.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

Procedure

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

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

2. Copy the `verify-topologysuffix.yml` Ansible playbook file provided by the **ansible-freeipa** package:

```
$ 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:
 - Indicate that the value of the `ipaadmin_password` variable is defined in the `secret.yml` Ansible vault file.
 - 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Verify topology suffix
    ipatopologysuffix:
      ipaadmin_password: "{{ ipaadmin_password }}"
      suffix: domain
      state: verified
```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory verify-
topologysuffix-copy.yml
```

Additional resources

- [Explaining Replication Agreements, Topology Suffixes, and Topology Segments](#)
- [/usr/share/doc/ansible-freeipa/README-topology.md](#)
- Sample playbooks in [/usr/share/doc/ansible-freeipa/playbooks/topology](#)

15.5. 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.

Follow this procedure to use an Ansible playbook to reinitialize the **domain** data on **replica.idm.example.com** from **server.idm.example.com**.

Prerequisites

- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.

- You have installed the **ansible-freeipa** package.
- The example assumes that in the `~/MyPlaybooks/` directory, you have created an **Ansible inventory file** with the fully-qualified domain name (FQDN) of the IdM server.
- The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password** and that you have access to a file that stores the password protecting the **secret.yml** file.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

Procedure

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

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

2. Copy the **reinitialize-topologysegment.yml** Ansible playbook file provided by the **ansible-freeipa** package:

```
$ 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:
 - Indicate that the value of the **ipaadmin_password** variable is defined in the **secret.yml** Ansible vault file.
 - 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

  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
    - name: Reinitialize topology segment
      ipatopologysegment:
        ipaadmin_password: "{{ ipaadmin_password }}"
        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. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory reinitialize-
topologysegment-copy.yml
```

Additional resources

- [Explaining Replication Agreements, Topology Suffixes, and Topology Segments](#)
- [/usr/share/doc/ansible-freeipa/README-topology.md](#)
- Sample playbooks in [/usr/share/doc/ansible-freeipa/playbooks/topology](#)

15.6. 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.

Follow this procedure 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

- You understand the recommendations for designing your IdM topology listed in [Connecting the replicas in a topology](#).
- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the **~/MyPlaybooks/** directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password** and that you have access to a file that stores the password protecting the **secret.yml** file.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

Procedure

1. Navigate to your **~/MyPlaybooks/** directory:

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

- Copy the **delete-topologysegment.yml** Ansible playbook file provided by the **ansible-freeipa** package:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/topology/delete-topologysegment.yml
delete-topologysegment-copy.yml
```

- Open the **delete-topologysegment-copy.yml** file for editing.
- Adapt the file by setting the following variables in the **ipatopologysegment** task section:
 - Indicate that the value of the **ipaadmin_password** variable is defined in the **secret.yml** Ansible vault file.
 - 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Delete topology segment
    ipatopologysegment:
      ipaadmin_password: "{{ ipaadmin_password }}"
      suffix: domain
      left: replica01.idm.example.com
      right: replica02.idm.example.com:
      state: absent
```

- Save the file.
- Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory delete-
topologysegment-copy.yml
```

Additional resources

- [Explaining Replication Agreements, Topology Suffixes, and Topology Segments](#)

- **`/usr/share/doc/ansible-freeipa/README-topology.md`**
- Sample playbooks in **`/usr/share/doc/ansible-freeipa/playbooks/topology`**

15.7. ADDITIONAL RESOURCES

- [Planning the replica topology](#).
- [Installing an IdM replica](#) .

CHAPTER 16. MANAGING IDM SERVERS BY USING ANSIBLE

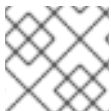
You can use **Red Hat Ansible Engine** to manage the servers in your Identity Management (IdM) topology. You can use the **server** module in the **ansible-freeipa** package to check the presence or absence of a server in the IdM topology. You can also hide any replica or make a replica visible.

The section contains the following topics:

- [Checking that an IdM server is present by using Ansible](#)
- [Ensuring that an IdM server is absent from an IdM topology by using Ansible](#)
- [Ensuring the absence of an IdM server despite hosting a last IdM server role](#)
- [Ensuring that an IdM server is absent but not necessarily disconnected from other IdM servers](#)
- [Ensuring that an existing IdM server is hidden using an Ansible playbook](#)
- [Ensuring that an existing IdM server is visible using an Ansible playbook](#)
- [Ensuring that an existing IdM server has an IdM DNS location assigned](#)
- [Ensuring that an existing IdM server has no IdM DNS location assigned](#)

16.1. CHECKING THAT AN IDM SERVER IS PRESENT BY USING ANSIBLE

You can use the **ipaserver ansible-freeipa** module in an Ansible playbook to verify that an Identity Management (IdM) server exists.



NOTE

The **ipaserver** Ansible module does not install the IdM server.

Prerequisites

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
 - The **SSH** connection from the control node to the IdM server defined in the inventory file is working correctly.

Procedure

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

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

- Copy the **server-present.yml** Ansible playbook file located in the **/usr/share/doc/ansible-freeipa/playbooks/server/** directory:

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

- Open the **server-present-copy.yml** file for editing.
- Adapt the file by setting the following variables in the **ipaserver** task section and save the file:
 - Set the **ipaadmin_password** variable to the password of the IdM **admin**.
 - Set the **name** variable to the **FQDN** of the server. The **FQDN** of the example server is **server123.idm.example.com**.

```
---
- name: Server present example
  hosts: ipaserver
  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
    - name: Ensure server server123.idm.example.com is present
      ipaserver:
        ipaadmin_password: "{{ ipaadmin_password }}"
        name: server123.idm.example.com
```

- Run the Ansible playbook and specify the playbook file and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory server-present-copy.yml
```

Additional resources

- [Installing an Identity Management server using an Ansible playbook](#)
- The **README-server.md** file in the **/usr/share/doc/ansible-freeipa/** directory
- The sample playbooks in the **/usr/share/doc/ansible-freeipa/playbooks/server** directory

16.2. ENSURING THAT AN IDM SERVER IS ABSENT FROM AN IDM TOPOLOGY BY USING ANSIBLE

Use an Ansible playbook to ensure an Identity Management (IdM) server does not exist in an IdM topology, even as a host.

In contrast to the **ansible-freeipa ipaserver** role, the **ipaserver** module used in this playbook does not uninstall IdM services from the server.

Prerequisites

- On the control node:

- You are using Ansible version 2.14 or later.
- You have installed the **ansible-freeipa** package.
- The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
- The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
 - The **SSH** connection from the control node to the IdM server defined in the inventory file is working correctly.

Procedure

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

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

2. Copy the **server-absent.yml** Ansible playbook file located in the `/usr/share/doc/ansible-freeipa/playbooks/server/` directory:

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

3. Open the **server-absent-copy.yml** file for editing.
4. Adapt the file by setting the following variables in the **ipaserver** task section and save the file:

- Set the **ipaadmin_password** variable to the password of the IdM **admin**.
- Set the **name** variable to the **FQDN** of the server. The **FQDN** of the example server is **server123.idm.example.com**.
- Ensure that the **state** variable is set to **absent**.

```
---
- name: Server absent example
  hosts: ipaserver
  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
    - name: Ensure server server123.idm.example.com is absent
      ipaserver:
        ipaadmin_password: "{{ ipaadmin_password }}"
        name: server123.idm.example.com
        state: absent
```

5. Run the Ansible playbook and specify the playbook file and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory server-absent-copy.yml
```

6. Make sure all name server (NS) DNS records pointing to **server123.idm.example.com** are deleted from your DNS zones. This applies regardless of whether you use integrated DNS managed by IdM or external DNS.

Additional resources

- [Uninstalling an IdM server](#)
- The **README-server.md** file in the `/usr/share/doc/ansible-freeipa/` directory
- Sample playbooks in the `/usr/share/doc/ansible-freeipa/playbooks/server` directory

16.3. ENSURING THE ABSENCE OF AN IDM SERVER DESPITE HOSTING A LAST IDM SERVER ROLE

You can use Ansible to ensure that an Identity Management (IdM) server is absent even if the last IdM service instance is running on the server. A certificate authority (CA), key recovery authority (KRA), or DNS server are all examples of IdM services.



WARNING

If you remove the last server that serves as a CA, KRA, or DNS server, you disrupt IdM functionality seriously. You can manually check which services are running on which IdM servers with the **ipa service-find** command. The principal name of a CA server is **dogtag/server_name/REALM_NAME**.

In contrast to the **ansible-freeipa ipaserver** role, the **ipaserver** module used in this playbook does not uninstall IdM services from the server.

Prerequisites

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
 - The **SSH** connection from the control node to the IdM server defined in the inventory file is working correctly.

Procedure

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

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

2. Copy the **server-absent-ignore-last-of-role.yml** Ansible playbook file located in the `/usr/share/doc/ansible-freeipa/playbooks/server/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/server/server-absent-ignore-last-of-role.yml server-absent-ignore-last-of-role-copy.yml
```

3. Open the **server-absent-ignore-last-of-role-copy.yml** file for editing.
4. Adapt the file by setting the following variables in the **ipaserver** task section and save the file:

- Set the **ipaadmin_password** variable to the password of the IdM **admin**.
- Set the **name** variable to the **FQDN** of the server. The **FQDN** of the example server is **server123.idm.example.com**.
- Ensure that the **ignore_last_of_role** variable is set to **true**.
- Set the **state** variable to **absent**.

```
---
- name: Server absent with last of role skip example
  hosts: ipaserver
  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
    - name: Ensure server "server123.idm.example.com" is absent with last of role skip
      ipaserver:
        ipaadmin_password: "{{ ipaadmin_password }}"
        name: server123.idm.example.com
        ignore_last_of_role: true
        state: absent
```

5. Run the Ansible playbook and specify the playbook file and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory server-absent-ignore-last-of-role-copy.yml
```

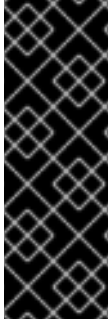
6. Make sure all name server (NS) DNS records that point to **server123.idm.example.com** are deleted from your DNS zones. This applies regardless of whether you use integrated DNS managed by IdM or external DNS.

Additional resources

- [Uninstalling an IdM server](#)
- The **README-server.md** file in the `/usr/share/doc/ansible-freeipa/` directory
- Sample playbooks in the `/usr/share/doc/ansible-freeipa/playbooks/server` directory

16.4. ENSURING THAT AN IDM SERVER IS ABSENT BUT NOT NECESSARILY DISCONNECTED FROM OTHER IDM SERVERS

If you are removing an Identity Management (IdM) server from the topology, you can keep its replication agreements intact with an Ansible playbook. The playbook also ensures that the IdM server does not exist in IdM, even as a host.



IMPORTANT

Ignoring a server's replication agreements when removing it is only recommended when the other servers are dysfunctional servers that you are planning to remove anyway. Removing a server that serves as a central point in the topology can split your topology into two disconnected clusters.

You can remove a dysfunctional server from the topology with the **ipa server-del** command.



NOTE

If you remove the last server that serves as a certificate authority (CA), key recovery authority (KRA), or DNS server, you seriously disrupt the Identity Management (IdM) functionality. To prevent this problem, the playbook makes sure these services are running on another server in the domain before it uninstalls a server that serves as a CA, KRA, or DNS server.

In contrast to the **ansible-freeipa ipaserver** role, the **ipaserver** module used in this playbook does not uninstall IdM services from the server.

Prerequisites

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the **~/MyPlaybooks/** directory, you have created an **Ansible inventory file** with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
 - The **SSH** connection from the control node to the IdM server defined in the inventory file is working correctly.

Procedure

1. Navigate to your **~/MyPlaybooks/** directory:

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

- Copy the **server-absent-ignore_topology_disconnect.yml** Ansible playbook file located in the **/usr/share/doc/ansible-freeipa/playbooks/server/** directory:

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

- Open the **server-absent-ignore_topology_disconnect-copy.yml** file for editing.
- Adapt the file by setting the following variables in the **ipaserver** task section and save the file:

- Set the **ipaadmin_password** variable to the password of the IdM **admin**.
- Set the **name** variable to the **FQDN** of the server. The **FQDN** of the example server is **server123.idm.example.com**.
- Ensure that the **ignore_topology_disconnect** variable is set to **true**.
- Ensure that the **state** variable is set to **absent**.

```
---
- name: Server absent with ignoring topology disconnects example
  hosts: ipaserver
  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
    - name: Ensure server "server123.idm.example.com" with ignoring topology disconnects
      ipaserver:
        ipaadmin_password: "{{ ipaadmin_password }}"
        name: server123.idm.example.com
        ignore_topology_disconnect: true
        state: absent
```

- Run the Ansible playbook and specify the playbook file and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory server-absent-
ignore_topology_disconnect-copy.yml
```

- Optional: Make sure all name server (NS) DNS records pointing to **server123.idm.example.com** are deleted from your DNS zones. This applies regardless of whether you use integrated DNS managed by IdM or external DNS.

Additional resources

- [Uninstalling an IdM server](#)
- The **README-server.md** file in the **/usr/share/doc/ansible-freeipa/** directory
- Sample playbooks in the **/usr/share/doc/ansible-freeipa/playbooks/server** directory.

16.5. ENSURING THAT AN EXISTING IDM SERVER IS HIDDEN USING AN ANSIBLE PLAYBOOK

Use the **ipaserver ansible-freeipa** module in an Ansible playbook to ensure that an existing Identity Management (IdM) server is hidden. Note that this playbook does not install the IdM server.

Prerequisites

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the `secret.yml` Ansible vault stores your `ipaadmin_password`.
- The target node, that is the node on which the `ansible-freeipa` module is executed, is part of the IdM domain as an IdM client, server or replica.
 - The **SSH** connection from the control node to the IdM server defined in the inventory file is working correctly.

Procedure

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

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

2. Copy the `server-hidden.yml` Ansible playbook file located in the `/usr/share/doc/ansible-freeipa/playbooks/server/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/server/server-hidden.yml server-hidden-copy.yml
```

3. Open the `server-hidden-copy.yml` file for editing.
4. Adapt the file by setting the following variables in the `ipaserver` task section and save the file:
 - Set the `ipaadmin_password` variable to the password of the IdM `admin`.
 - Set the `name` variable to the **FQDN** of the server. The **FQDN** of the example server is `server123.idm.example.com`.
 - Ensure that the `hidden` variable is set to **True**.

```
---
- name: Server hidden example
  hosts: ipaserver
  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
    - name: Ensure server server123.idm.example.com is hidden
      ipaserver:
        ipaadmin_password: "{{ ipaadmin_password }}"
        name: server123.idm.example.com
        hidden: True
```

5. Run the Ansible playbook and specify the playbook file and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory server-hidden-copy.yml
```

Additional resources

- [Installing an Identity Management server using an Ansible playbook](#)
- [The hidden replica mode](#)
- The **README-server.md** file in the `/usr/share/doc/ansible-freeipa/` directory
- Sample playbooks in the `/usr/share/doc/ansible-freeipa/playbooks/server` directory

16.6. ENSURING THAT AN EXISTING IDM SERVER IS VISIBLE BY USING AN ANSIBLE PLAYBOOK

Use the **ipaserver ansible-freeipa** module in an Ansible playbook to ensure that an existing Identity Management (IdM) server is visible. Note that this playbook does not install the IdM server.

Prerequisites

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
 - The **SSH** connection from the control node to the IdM server defined in the inventory file is working correctly.

Procedure

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

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

2. Copy the **server-not-hidden.yml** Ansible playbook file located in the `/usr/share/doc/ansible-freeipa/playbooks/server/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/server/server-not-hidden.yml server-not-hidden-copy.yml
```

3. Open the **server-not-hidden-copy.yml** file for editing.
4. Adapt the file by setting the following variables in the **ipaserver** task section and save the file:

- Set the **ipaadmin_password** variable to the password of the IdM **admin**.
- Set the **name** variable to the **FQDN** of the server. The **FQDN** of the example server is **server123.idm.example.com**.
- Ensure that the **hidden** variable is set to **no**.

```
---
- name: Server not hidden example
  hosts: ipaserver
  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
    - name: Ensure server server123.idm.example.com is not hidden
      ipaserver:
        ipaadmin_password: "{{ ipaadmin_password }}"
        name: server123.idm.example.com
        hidden: no
```

5. Run the Ansible playbook and specify the playbook file and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory server-not-hidden-copy.yml
```

Additional resources

- [Installing an Identity Management server using an Ansible playbook](#)
- [The hidden replica mode](#)
- The **README-server.md** file in the **/usr/share/doc/ansible-freeipa/** directory
- The sample playbooks in the **/usr/share/doc/ansible-freeipa/playbooks/server** directory

16.7. ENSURING THAT AN EXISTING IDM SERVER HAS AN IDM DNS LOCATION ASSIGNED

Use the **ipaserver ansible-freeipa** module in an Ansible playbook to ensure that an existing Identity Management (IdM) server is assigned a specific IdM DNS location.

Note that the **ipaserver** Ansible module does not install the IdM server.

Prerequisites

- You know the IdM **admin** password.
- The IdM DNS location exists. The example location is **germany**.
- You have **root** access to the server. The example server is **server123.idm.example.com**.
- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.

- The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
- The example assumes that the `secret.yml` Ansible vault stores your `ipaadmin_password`.
- The target node, that is the node on which the `ansible-freeipa` module is executed, is part of the IdM domain as an IdM client, server or replica.
 - The **SSH** connection from the control node to the IdM server defined in the inventory file is working correctly.

Procedure

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

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

2. Copy the `server-location.yml` Ansible playbook file located in the `/usr/share/doc/ansible-freeipa/playbooks/server/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/server/server-location.yml server-location-copy.yml
```

3. Open the `server-location-copy.yml` file for editing.
4. Adapt the file by setting the following variables in the `ipaserver` task section and save the file:
 - Set the `ipaadmin_password` variable to the password of the IdM `admin`.
 - Set the `name` variable to `server123.idm.example.com`.
 - Set the `location` variable to `germany`.

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

```
---
- name: Server enabled example
  hosts: ipaserver
  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
    - name: Ensure server server123.idm.example.com with location "germany" is present
      ipaserver:
        ipaadmin_password: "{{ ipaadmin_password }}"
        name: server123.idm.example.com
        location: germany
```

5. Run the Ansible playbook and specify the playbook file and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory server-location-copy.yml
```

6. Connect to `server123.idm.example.com` as `root` using **SSH**:

```
ssh root@server123.idm.example.com
```

7. Restart the **named-pkcs11** service on the server for the updates to take effect immediately:

```
[root@server123.idm.example.com ~]# systemctl restart named-pkcs11
```

Additional resources

- [Installing an Identity Management server using an Ansible playbook](#)
- [Using Ansible to ensure an IdM location is present](#)
- The **README-server.md** file in the `/usr/share/doc/ansible-freeipa/` directory
- Sample playbooks in the `/usr/share/doc/ansible-freeipa/playbooks/server` directory

16.8. ENSURING THAT AN EXISTING IDM SERVER HAS NO IDM DNS LOCATION ASSIGNED

Use the **ipaserver ansible-freeipa** module in an Ansible playbook to ensure that an existing Identity Management (IdM) server has no IdM DNS location assigned to it. Do not assign a DNS location to servers that change geographical location frequently. Note that the playbook does not install the IdM server.

Prerequisites

- You know the IdM **admin** password.
- You have **root** access to the server. The example server is **server123.idm.example.com**.
- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
 - The **SSH** connection from the control node to the IdM server defined in the inventory file is working correctly.

Procedure

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

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

2. Copy the **server-no-location.yml** Ansible playbook file located in the `/usr/share/doc/ansible-freeipa/playbooks/server/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/server/server-no-location.yml server-no-location-copy.yml
```

3. Open the **server-no-location-copy.yml** file for editing.
4. Adapt the file by setting the following variables in the **ipaserver** task section and save the file:
 - Set the **ipaadmin_password** variable to the password of the IdM **admin**.
 - Set the **name** variable to **server123.idm.example.com**.
 - Ensure that the **location** variable is set to **""**.

```
---
- name: Server no location example
  hosts: ipaserver

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure server server123.idm.example.com is present with no location
    ipaserver:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: server123.idm.example.com
      location: ""
```

5. Run the Ansible playbook and specify the playbook file and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory server-no-location-copy.yml
```

6. Connect to **server123.idm.example.com** as **root** using **SSH**:

```
ssh root@server123.idm.example.com
```

7. Restart the **named-pkcs11** service on the server for the updates to take effect immediately:

```
[root@server123.idm.example.com ~]# systemctl restart named-pkcs11
```

Additional resources

- [Installing an Identity Management server using an Ansible playbook](#)
- [Using Ansible to manage DNS locations in IdM](#)
- The **README-server.md** file in the **/usr/share/doc/ansible-freeipa/** directory
- Sample playbooks in the **/usr/share/doc/ansible-freeipa/playbooks/server** directory

CHAPTER 17. MANAGING HOSTS USING ANSIBLE PLAYBOOKS

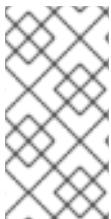
Ansible is an automation tool used to configure systems, deploy software, and perform rolling updates. Ansible includes support for Identity Management (IdM), and you can use Ansible modules to automate host management.

17.1. ENSURING THE PRESENCE OF AN IDM HOST ENTRY WITH FQDN USING ANSIBLE PLAYBOOKS

Follow this procedure to ensure 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.
- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an **Ansible inventory file** with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Host host01.idm.example.com present
    ipahost:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: host01.idm.example.com
      state: present
      force: true
```

3. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -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

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.

17.2. ENSURING THE PRESENCE OF AN IDM HOST ENTRY WITH DNS INFORMATION USING ANSIBLE PLAYBOOKS

Follow this procedure to ensure 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.
- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the `secret.yml` Ansible vault stores your `ipaadmin_password`.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

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

  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
    - name: Ensure host01.idm.example.com is present
      ipahost:
        ipadmin_password: "{{ ipadmin_password }}"
        name: host01.idm.example.com
```

```

description: Example host
ip_address: 192.168.0.123
locality: Lab
ns_host_location: Lab
ns_os_version: RHEL 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 --vault-password-file=password_file -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

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: RHEL 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.

17.3. 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. Follow this procedure 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.
- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

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: true** and **force: true** 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

  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
    - name: Hosts host01.idm.example.com and host02.idm.example.com present with random passwords
      ipahost:
        ipaadmin_password: "{{ ipaadmin_password }}"
      hosts:
```



```
- name: host01.idm.example.com
  random: true
  force: true
- name: host02.idm.example.com
  random: true
  force: true
register: ipahost
```

3. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -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": "0HolRvjUdH0Ycbf6uYdWTxH"},
"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

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.

17.4. ENSURING THE PRESENCE OF AN IDM HOST ENTRY WITH MULTIPLE IP ADDRESSES USING ANSIBLE PLAYBOOKS

Follow this procedure 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.
- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the `secret.yml` Ansible vault stores your `ipaadmin_password`.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

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

  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
    - name: Ensure host101.example.com IP addresses present
      ipahost:
        ipadmin_password: "{{ ipadmin_password }}"
        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: true
```

3. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -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

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.

17.5. ENSURING THE ABSENCE OF AN IDM HOST ENTRY USING ANSIBLE PLAYBOOKS

Follow this procedure 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: true** 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Host host01.idm.example.com absent
    ipahost:
      ipadmin_password: "{{ ipadmin_password }}"
      name: host01.idm.example.com
      updatedns: true
      state: absent
```

3. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -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

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.

17.6. ADDITIONAL RESOURCES

- See the `/usr/share/doc/ansible-freeipa/README-host.md` Markdown file.
- See the additional playbooks in the `/usr/share/doc/ansible-freeipa/playbooks/host` directory.

CHAPTER 18. MANAGING HOST GROUPS USING ANSIBLE PLAYBOOKS

Learn more about [host groups in Identity Management](#) (IdM) and using Ansible to perform operations involving host groups in Identity Management (IdM).

18.1. 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.

18.2. ENSURING THE PRESENCE OF IDM HOST GROUPS USING ANSIBLE PLAYBOOKS

Follow this procedure 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.

- The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
- The example assumes that the `secret.yml` Ansible vault stores your `ipaadmin_password`.
- The target node, that is the node on which the `ansible-freeipa` module is executed, is part of the IdM domain as an IdM client, server or replica.

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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  # Ensure host-group databases is present
  - ipahostgroup:
      ipaadmin_password: "{{ ipaadmin_password }}"
      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 --vault-password-file=password_file -v -i
path_to_inventory_directory/inventory.file path_to_playbooks_directory/ensure-
hostgroup-is-present.yml
```

Verification

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.

18.3. ENSURING THE PRESENCE OF HOSTS IN IDM HOST GROUPS USING ANSIBLE PLAYBOOKS

Follow this procedure 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  # Ensure host-group databases is present
  - ipahostgroup:
      ipaadmin_password: "{{ ipaadmin_password }}"
      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 --vault-password-file=password_file -v -i
path_to_inventory_directory/inventory.file path_to_playbooks_directory/ensure-hosts-
or-hostgroups-are-present-in-hostgroup.yml

```

Verification

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.

18.4. NESTING IDM HOST GROUPS USING ANSIBLE PLAYBOOKS

Follow this procedure to ensure the presence of nested host groups in Identity Management (IdM) host groups using Ansible playbooks.

Prerequisites

- You know the IdM administrator password.
- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the `secret.yml` Ansible vault stores your `ipaadmin_password`.
- The target node, that is the node on which the `ansible-freeipa` module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  # Ensure hosts and hostgroups are present in existing databases hostgroup
  - ipahostgroup:
      ipaadmin_password: "{{ ipaadmin_password }}"
      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 --vault-password-file=password_file -v -i
path_to_inventory_directory/inventory.file path_to_playbooks_directory/ensure-hosts-
or-hostgroups-are-present-in-hostgroup.yml
```

Verification

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.

18.5. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure member manager user example_member is present for group_name
    ipahostgroup:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: group_name
      membermanager_user: example_member

  - name: Ensure member manager group project_admins is present for group_name
    ipahostgroup:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: group_name
      membermanager_group: project_admins
```

3. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i
path_to_inventory_directory/inventory.file path_to_playbooks_directory/add-member-
managers-host-groups.yml
```

Verification

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 on your system.

18.6. ENSURING THE ABSENCE OF HOSTS FROM IDM HOST GROUPS USING ANSIBLE PLAYBOOKS

Follow this procedure 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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
```

```
vars_files:
- /home/user_name/MyPlaybooks/secret.yml
tasks:
# Ensure host-group databases is absent
- ipahostgroup:
  ipadmin_password: "{{ ipadmin_password }}"
  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 --vault-password-file=password_file -v -i
path_to_inventory_directory/inventory.file path_to_playbooks_directory/ensure-hosts-
or-hostgroups-are-absent-in-hostgroup.yml
```

Verification

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.

18.7. ENSURING THE ABSENCE OF NESTED HOST GROUPS FROM IDM HOST GROUPS USING ANSIBLE PLAYBOOKS

Follow this procedure 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the `secret.yml` Ansible vault stores your `ipaadmin_password`.
- The target node, that is the node on which the `ansible-freeipa` module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  # Ensure hosts and hostgroups are absent in existing databases hostgroup
  - ipahostgroup:
    ipaadmin_password: "{{ ipaadmin_password }}"
    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 --vault-password-file=password_file -v -i
path_to_inventory_directory/inventory.file path_to_playbooks_directory/ensure-hosts-
or-hostgroups-are-absent-in-hostgroup.yml
```

Verification

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.

18.8. ENSURING THE ABSENCE OF IDM HOST GROUPS USING ANSIBLE PLAYBOOKS

Follow this procedure 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the **~/MyPlaybooks/** directory, you have created an **Ansible inventory file** with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.

- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - Ensure host-group databases is absent
    ipahostgroup:
      ipaadmin_password: "{{ ipaadmin_password }}"
      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 --vault-password-file=password_file -v -i
path_to_inventory_directory/inventory.file path_to_playbooks_directory/ensure-
hostgroup-is-absent.yml
```

Verification

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.

18.9. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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

  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
    - name: Ensure member manager host and host group members are absent for
      group_name
      ipahostgroup:
        ipaadmin_password: "{{ ipaadmin_password }}"
        name: group_name
        membermanager_user: example_member
        membermanager_group: project_admins
        action: member
        state: absent
```

3. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i  
path_to_inventory_directory/inventory.file path_to_playbooks_directory/ensure-  
member-managers-host-groups-are-absent.yml
```

Verification

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 on your system.

CHAPTER 19. DEFINING IDM PASSWORD POLICIES

Password policies help to reduce the risk of someone discovering and misusing a user's password. You can add Identity Management (IdM) password policies in the IdM WebUI or the CLI. Additionally, you can add a new password policy in IdM using an Ansible playbook.

19.1. 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.

19.2. PASSWORD POLICIES IN IDM

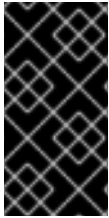
Passwords are the most common way for Identity Management (IdM) users to authenticate to the IdM Kerberos domain. Password policies define the requirements that these IdM user passwords must meet. The IdM password policy is set in the underlying LDAP directory, but the Kerberos Key Distribution Center (KDC) enforces the password policy.

[Password policy attributes](#) lists the attributes you can use to define a password policy in IdM.

Table 19.1. 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. The default value is 90 days. Note that if the attribute is set to 0, the password never expires.	Max lifetime = 180 User passwords are valid only for 180 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 characters.</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 the Red Hat Knowledgebase solution [What characters are valid in a password?](#) .

19.3. PASSWORD POLICY PRIORITIES IN IDM

Password policies help reduce the risk of someone discovering and misusing a user's password. The default password policy is the *global password policy*. You can also create additional group password policies. The global policy rules apply to all users without a group password policy. Group password policies apply to all members of the corresponding user group.

Note that only one password policy can be in effect at a time for any user. If a user has multiple password policies assigned, one of them takes precedence based on priority according to the following rules:

- Every group password policy has a priority set. The lower the value, the higher the policy's priority. The lowest supported value is **0**.
- If multiple password policies are applicable to a user, the policy with the lowest priority value takes precedence. All rules defined in other policies are ignored.
- The password policy with the lowest priority value applies to all password policy attributes, even the attributes that are not defined in the policy.

The global password policy does not have a priority value set. It serves as a fallback policy when no group policy is set for a user. The global policy can never take precedence over a group policy.

You can use the **ipa pwpolicy-show --user=user_name** command to check which policy is currently in effect for a particular user.

19.4. ENSURING THE PRESENCE OF A PASSWORD POLICY IN IDM USING AN ANSIBLE PLAYBOOK

Follow this procedure 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 cannot define a password policy for an individual user.

Prerequisites

- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.

- You have installed the **ansible-freeipa** package.
- The example assumes that in the `~/MyPlaybooks/` directory, you have created an **Ansible inventory file** with the fully-qualified domain name (FQDN) of the IdM server.
- The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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

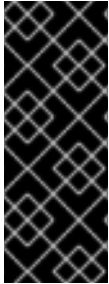
  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure presence of pwpolicy for group ops
    ipapwpolicy:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: ops
      minlife: 7
      maxlife: 49
      history: 5
      priority: 1
      lockouttime: 300
      minlength: 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 --vault-password-file=password_file -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

- See the **README-pwpolicy.md** file in the `/usr/share/doc/ansible-freeipa/` directory.
- See [Password policy priorities in IdM](#) .

19.5. ADDING A NEW PASSWORD POLICY IN IDM USING THE WEBUI OR THE CLI

Password policies help reduce the risk of someone discovering and misusing a user's password. The default password policy is the *global password policy*. You can also create additional group password policies.

19.5.1. Adding a new password policy in the IdM WebUI

Password policies help reduce the risk of someone discovering and misusing a user's password. The default password policy is the *global password policy*. You can also create additional group password policies.

Prerequisites

- A user group to which the policy applies.
- A priority assigned to the policy

Procedure

1. Log in to the IdM Web UI.
For details, see [Accessing the IdM Web UI in a web browser](#) .
2. Select **Policy>Password Policies**.
3. Click **Add**.
4. Define the user group and priority.
5. Click **Add** to confirm.

To configure the attributes of the new password policy, see [Password policies in IdM](#) .

Additional resources

- See [Password policy priorities in IdM](#) .

19.5.2. Adding a new password policy in the IdM CLI

Password policies help reduce the risk of someone discovering and misusing a user's password. The default password policy is the *global password policy*. You can also create additional group password policies.

Prerequisites

- A user group to which the policy applies.
- A priority assigned to the policy

Procedure

1. Open terminal and connect to the IdM server.
2. Use the **ipa pwpolicy-add** command. Specify the user group and priority:

```
$ ipa pwpolicy-add
Group: group_name
Priority: priority_level
```

3. Optional: Use the **ipa pwpolicy-find** command to verify that the policy has been successfully added:

```
$ ipa pwpolicy-find
```

To configure the attributes of the new password policy, see [Password policies in IdM](#) .

Additional resources

- See [Password policy priorities in IdM](#) .

19.6. 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:

--maxrepeat

Specifies the maximum acceptable number of same consecutive characters in the new password.

--maxsequence

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.

--dictcheck

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.

--usercheck

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, RHEL 8, and RHEL 9 servers, you can enforce the additional password policy settings only on servers running on RHEL 8.4 and later. If a user is logged in to an IdM client and the IdM client is communicating with an IdM server running on RHEL 8.3 or earlier, then the new password policy requirements set by the system administrator are not applied. To ensure consistent behavior, upgrade or update all servers to RHEL 8.4 and later.

Additional resources:

- [Applying additional password policies to an IdM group](#)
- **pwquality(3)** man page on your system

19.7. APPLYING ADDITIONAL PASSWORD POLICY OPTIONS TO AN IDM GROUP

Follow this procedure 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 the **managers** group name.
 - c. Click **Add**.
 - d. On 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. From the **Actions** menu, click **Reset Password**.
 - d. Enter a temporary password for the user.
4. Try to obtain a Kerberos ticket-granting ticket (TGT) for the **test_user**:

- a. On the command line, run the following command:

```
$ kinit test_user
```

- b. Enter the temporary password.
- c. 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:
Password change rejected: Password not changed.
Unspecified password quality failure while trying to change password.
Please try again.
```

- d. 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:

```

- e. 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](#)

19.8. USING AN ANSIBLE PLAYBOOK TO APPLY ADDITIONAL PASSWORD POLICY OPTIONS TO AN IDM GROUP

You can use an Ansible playbook to apply additional password policy options to strengthen the password policy requirements for a specific IdM group. You can use the **maxrepeat**, **maxsequence**, **dictcheck** and **usercheck** password policy options for this purpose. The example describes how to set the following requirements for the **managers** group:

- A user's new password does not contain the user's respective user name.
- The password contains no more than two identical characters in succession.
- Any monotonic character sequences in the passwords are not longer than 3 characters. This means that the system does not accept a password with a sequence such as **1234** or **abcd**.

Prerequisites

- You have configured your Ansible control node to meet the following requirements:

- You are using Ansible version 2.14 or later.
- You have installed the [ansible-freeipa](#) package on the Ansible controller.
- You have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server in the `~/MyPlaybooks/` directory.
- You have stored your `ipaadmin_password` in the `secret.yml` Ansible vault.
- The group for which you are ensuring the presence of a password policy exists in IdM.

Procedure

1. Create your Ansible playbook file `manager_pwpolicy_present.yml` that defines the password policy whose presence you want to ensure. To simplify this step, copy and modify the following example:

```
---
- name: Tests
  hosts: ipaserver

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure presence of usercheck and maxrepeat pwpolicy for group managers
    ipapwpolicy:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: managers
      usercheck: True
      maxrepeat: 2
      maxsequence: 3
```

2. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i
  path_to_inventory_directory/inventory.file
  path_to_playbooks_directory/manager_pwpolicy_present.yml
```

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. On 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. From the **Actions** menu, click **Reset Password**.
 - d. Enter a temporary password for the user.
 4. Try to obtain a Kerberos ticket-granting ticket (TGT) for the **test_user**:
 - a. On the command line, enter the following command:

```
$ kinit test_user
```
 - b. Enter the temporary password.
 - c. 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:  
Password change rejected: Password not changed.  
Unspecified password quality failure while trying to change password.  
Please try again.
```
 - d. 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:
```
 - e. The system informs you that the entered password was rejected. Enter a password that contains a monotonic character sequence longer than 3 characters. Examples of such sequences include **1234** and **fedc**:

```
Password change rejected: Password not changed.  
Unspecified password quality failure while trying to change password.  
Please try again.
```

```
Enter new password:  
Enter it again:
```

-
- f. 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. Verify that you have obtained a TGT, which is only possible after having entered a valid password:

```

$ 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

```

Additional resources

- [Additional password policies in IdM](#)
- `/usr/share/doc/ansible-freeipa/README-pwpolicy.md`
- `/usr/share/doc/ansible-freeipa/playbooks/pwpolicy`

CHAPTER 20. GRANTING SUDO ACCESS TO AN IDM USER ON AN IDM CLIENT

Learn more about granting **sudo** access to users in Identity Management.

20.1. 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

You can create a **central sudo rule** for an IdM client using the command line (CLI) and the IdM Web UI.

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

- [Managing sudo access](#)

20.2. 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
-----
```

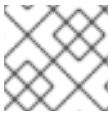
7. Optional: Define the validity of the **idm_user_reboot** rule:

- a. To define the time at which a **sudo** rule starts to be valid, use the **ipa sudorule-mod sudo_rule_name** command with the **--setattr sudonotbefore=DATE** option. The *DATE* value must follow the **yyyymmddHHMMSSZ** format, with seconds specified explicitly. For example, to set the start of the validity of the **idm_user_reboot** rule to 31 December 2025 12:34:00, enter:

```
[root@idmclient ~]# ipa sudorule-mod idm_user_reboot --setattr
sudonotbefore=20251231123400Z
```

- b. To define the time at which a sudo rule stops being valid, use the **--setattr sudonotafter=DATE** option. For example, to set the end of the **idm_user_reboot** rule validity to 31 December 2026 12:34:00, enter:

```
[root@idmclient ~]# ipa sudorule-mod idm_user_reboot --setattr
sudonotafter=20261231123400Z
```



NOTE

Propagating the changes from the server to the client can take a few minutes.

Verification

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:
```

20.3. GRANTING SUDO ACCESS TO AN AD USER ON AN IDM CLIENT USING THE CLI

Identity Management (IdM) system administrators can use IdM user groups to set access permissions, host-based access control, **sudo** rules, and other controls on IdM users. IdM user groups grant and restrict access to IdM domain resources.

You can add both Active Directory (AD) *users* and AD *groups* to IdM user groups. To do that:

1. Add the AD users or groups to a *non-POSIX* external IdM group.
2. Add the non-POSIX external IdM group to an IdM *POSIX* group.

You can then manage the privileges of the AD users by managing the privileges of the POSIX group. For example, you can grant **sudo** access for a specific command to an IdM POSIX user group on a specific IdM host.

It is also possible to add AD user groups as members to IdM external groups. This might make it easier to define policies for Windows users, by keeping the user and group management within the single AD realm.



IMPORTANT

Do **not** use ID overrides of AD users for SUDO rules in IdM. ID overrides of AD users represent only POSIX attributes of AD users, not AD users themselves.

You can add ID overrides as group members. However, you can only use this functionality to manage IdM resources in the IdM API. The possibility to add ID overrides as group members is not extended to POSIX environments and you therefore cannot use it for membership in **sudo** or host-based access control (HBAC) rules.

Follow this procedure to create the **ad_users_reboot sudo** rule to grant the **administrator@ad-domain.com** AD user the permission to run the **/usr/sbin/reboot** command on the **idmclient** IdM host, which is normally reserved for the **root** user. **administrator@ad-domain.com** is a member of the **ad_users_external** non-POSIX group, which is, in turn, a member of the **ad_users** POSIX group.

Prerequisites

- You have obtained the IdM **admin** Kerberos ticket-granting ticket (TGT).
- A cross-forest trust exists between the IdM domain and the **ad-domain.com** AD domain.
- No local **administrator** account is present on the **idmclient** host: the **administrator** user is not listed in the local **/etc/passwd** file.

Procedure

1. Create the **ad_users** group that contains the **ad_users_external** group with the **administrator@ad-domain** member:
 - a. Optional: Create or select a corresponding group in the AD domain to use to manage AD users in the IdM realm. You can use multiple AD groups and add them to different groups on the IdM side.

- b. Create the **ad_users_external** group and indicate that it contains members from outside the IdM domain by adding the **--external** option:

```
[root@ipaserver ~]# ipa group-add --desc='AD users external map'
ad_users_external --external
-----
Added group "ad_users_external"
-----
Group name: ad_users_external
Description: AD users external map
```



NOTE

Ensure that the external group that you specify here is an AD security group with a **global** or **universal** group scope as defined in the [Active Directory security groups](#) document. For example, the **Domain users** or **Domain admins** AD security groups cannot be used because their group scope is **domain local**.

- c. Create the **ad_users** group:

```
[root@ipaserver ~]# ipa group-add --desc='AD users' ad_users
-----
Added group "ad_users"
-----
Group name: ad_users
Description: AD users
GID: 129600004
```

- d. Add the **administrator@ad-domain.com** AD user to **ad_users_external** as an external member:

```
[root@ipaserver ~]# ipa group-add-member ad_users_external --external
"administrator@ad-domain.com"
[member user]:
[member group]:
Group name: ad_users_external
Description: AD users external map
External member: S-1-5-21-3655990580-1375374850-1633065477-513
-----
Number of members added 1
-----
```

The AD user must be identified by a fully-qualified name, such as **DOMAIN\user_name** or **user_name@DOMAIN**. The AD identity is then mapped to the AD SID for the user. The same applies to adding AD groups.

- e. Add **ad_users_external** to **ad_users** as a member:

```
[root@ipaserver ~]# ipa group-add-member ad_users --groups ad_users_external
Group name: ad_users
Description: AD users
GID: 129600004
Member groups: ad_users_external
```

```
-----
Number of members added 1
-----
```

2. Grant the members of **ad_users** the permission to run **/usr/sbin/reboot** on the **idmclient** host:

- a. 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
```

- b. Create a **sudo** rule named **ad_users_reboot**:

```
[root@idmclient ~]# ipa sudorule-add ad_users_reboot
-----
Added Sudo Rule "ad_users_reboot"
-----
Rule name: ad_users_reboot
Enabled: True
```

- c. Add the **/usr/sbin/reboot** command to the **ad_users_reboot** rule:

```
[root@idmclient ~]# ipa sudorule-add-allow-command ad_users_reboot --sudocmds
'/usr/sbin/reboot'
Rule name: ad_users_reboot
Enabled: True
Sudo Allow Commands: /usr/sbin/reboot
-----
Number of members added 1
-----
```

- d. Apply the **ad_users_reboot** rule to the IdM **idmclient** host:

```
[root@idmclient ~]# ipa sudorule-add-host ad_users_reboot --hosts
idmclient.idm.example.com
Rule name: ad_users_reboot
Enabled: True
Hosts: idmclient.idm.example.com
Sudo Allow Commands: /usr/sbin/reboot
-----
Number of members added 1
-----
```

- e. Add the **ad_users** group to the **ad_users_reboot** rule:

```
[root@idmclient ~]# ipa sudorule-add-user ad_users_reboot --groups ad_users
Rule name: ad_users_reboot
Enabled: TRUE
User Groups: ad_users
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

1. Log in to the **idmclient** host as **administrator@ad-domain.com**, an indirect member of the **ad_users** group:

```
$ ssh administrator@ad-domain.com@ipacient
```

Password:

2. Optional: Display the **sudo** commands that **administrator@ad-domain.com** is allowed to execute:

```
[administrator@ad-domain.com@idmclient ~]$ sudo -l
Matching Defaults entries for administrator@ad-domain.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 administrator@ad-domain.com may run the following commands on idmclient:
    (root) /usr/sbin/reboot
```

3. Reboot the machine using **sudo**. Enter the password for **administrator@ad-domain.com** when prompted:

```
[administrator@ad-domain.com@idmclient ~]$ sudo /usr/sbin/reboot
[sudo] password for administrator@ad-domain.com:
```

Additional resources

- [Active Directory users and Identity Management groups](#)
- [Include users and groups from a trusted Active Directory domain into SUDO rules](#)

20.4. 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, 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**.
 - 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 **Users** section, click **Add** to open the **Add users into sudo rule "idm_user_reboot"** dialog box.
 - iii. In the **Available** column, check the **idm_user** checkbox, and click the arrow to 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 **Hosts** section, click **Add** to open the **Add hosts into sudo rule "idm_user_reboot"** dialog box.
 - iii. In **Available** column, check the **idmclient.idm.example.com** checkbox, and click the arrow to move it to the **Prospective** column.

- iv. Click **Add**.
- g. Specify the commands:
 - i. In the **Run Commands** section, check the **Specified Commands and Groups** radio button.
 - ii. In the **Sudo Allow Commands** section, click **Add** to open the **Add allow sudo commands into sudo rule "idm_user_reboot"** dialog box.
 - iii. In the **Available** column, check the **/usr/sbin/reboot** checkbox, and click the arrow to move it to the **Prospective** column.
 - iv. Click **Add** to return to the **idm_sudo_reboot** page.
- h. 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

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.

20.5. 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 **sudorule-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 **run_third-party-app_report** 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

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.
```

20.6. 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**.
 - 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**.
- 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** section, click **Add** to open the **Add users into sudo rule "run_third-party-app_report"** dialog box.
 - iii. 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 **Hosts** section, click **Add** to open the **Add hosts into sudo rule "run_third-party-app_report"** dialog box.
 - iii. 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 **Run Commands** section, check the **Specified Commands and Groups** radio button.
 - ii. In the **Sudo Allow Commands** section, click **Add** to open the **Add allow sudo commands into sudo rule "run_third-party-app_report"** dialog box.
 - iii. In the **Available** column, check the **/opt/third-party-app/bin/report** checkbox, and move it to the **Prospective** column.
 - 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** section, click **Add** to open the **Add RunAs users into sudo rule "run_third-party-app_report"** dialog box.
 - iii. Enter the **thirdpartyapp** service account in the **External** box and move it to the **Prospective** column.
 - 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.



NOTE

Propagating the changes from the server to the client can take a few minutes.

Verification

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.
```

20.7. ENABLING GSSAPI AUTHENTICATION FOR SUDO ON AN IDM CLIENT

Enable Generic Security Service Application Program Interface (GSSAPI) authentication on an IdM client for the **sudo** and **sudo -i** commands via the **pam_sss_gss.so** PAM module. With this configuration, IdM users can 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.
- 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. On RHEL 9.2 or later:
 - a. Optional: Determine if you have selected the **sss authselect** profile:

```
# authselect current
Profile ID: sssd
```

- b. If the **sss authselect** profile is selected, enable GSSAPI authentication:

```
# authselect enable-feature with-gssapi
```

- c. If the **sss authselect** profile is not selected, select it and enable GSSAPI authentication:

```
# authselect select sssd with-gssapi
```

6. On RHEL 9.1 or earlier:
 - a. Open the **/etc/pam.d/sudo** PAM configuration file.
 - b. 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
```

- c. Save and close the **/etc/pam.d/sudo** file.

Verification

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
```

- Optional: If you do not have Kerberos credentials for the **idm_user** account, delete 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:
```

- Reboot the machine using **sudo**, without specifying a password.

```
[idm_user@idmclient ~]$ sudo /usr/sbin/reboot
```

Additional resources

- The GSSAPI entry in the [IdM terminology](#) listing
- [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)** and **sssd.conf (5)** man pages on your system

20.8. ENABLING GSSAPI AUTHENTICATION AND ENFORCING KERBEROS AUTHENTICATION INDICATORS FOR SUDO ON AN IDM CLIENT

Enable Generic Security Service Application Program Interface (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.

- 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/<idm_domain_name>]** section.

```
[domain/<idm_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. On RHEL 9.2 or later:
 - a. Determine if you have selected the **sss authselect** profile:

```
# authselect current
Profile ID: sssd
```

- b. Optional: Select the **sss authselect** profile:

```
# authselect select sssd
```

- c. Enable GSSAPI authentication:

```
# authselect enable-feature with-gssapi
```

- d. Configure the system to authenticate only users with smart cards:

```
# authselect with-smartcard-required
```

6. On RHEL 9.1 or earlier:
 - a. Open the **/etc/pam.d/sudo** PAM configuration file.
 - b. 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
```

- c. Save and close the **/etc/pam.d/sudo** file.
- d. Open the **/etc/pam.d/sudo-i** PAM configuration file.

- e. 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
```

- f. Save and close the **/etc/pam.d/sudo-i** file.

Verification

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

- [SSSD options controlling GSSAPI authentication for PAM services](#)
- The GSSAPI entry in the [IdM terminology](#) listing
- [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)** and **sssd.conf (5)** man pages on your system

20.9. SSSD OPTIONS CONTROLLING GSSAPI AUTHENTICATION FOR PAM SERVICES

You can use the following options for the `/etc/sssd/sssd.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](#)

20.10. 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.

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/sss.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

```

20.11. 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the **~/MyPlaybooks/** directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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, see link: [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

  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
    # Ensure sudo command is present
    - ipasudocmd:
        ipadmin_password: "{{ ipadmin_password }}"
        name: /usr/sbin/reboot
        state: present
```

- b. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -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

  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
    # Ensure a sudorule is present granting idm_user the permission to run /usr/sbin/reboot
    on idmclient
    - ipasudorule:
        ipadmin_password: "{{ ipadmin_password }}"
```

```

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

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 resources

- See the **README-sudocmd.md**, **README-sudocmdgroup.md**, and **README-sudorule.md** files in the **/usr/share/doc/ansible-freeipa/** directory.

20.12. MANAGING MULTIPLE IDM SUDO RULES IN A SINGLE ANSIBLE TASK

Using the **sudorules** option available in the **ansible-freeipa** module, you can ensure the presence or absence of multiple Identity Management (IdM) **sudo** rules in a single Ansible task. Using the option, you can thus define your **sudo** rules more easily, and execute them more efficiently.

Prerequisites

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You are using RHEL 9.6 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the **~/MyPlaybooks/** directory, you have created an **Ansible inventory file** with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password** and that you have access to a file that stores the password protecting the **secret.yml** file.

- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

The example further assumes the following:

- The **user01** and **user02** users exist in IdM.
- The **usergroup01** user group exists in IdM.
- The **hostgroup01** and **hostgroup02** host groups exist in IdM.

Procedure

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

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

2. Create an **ensure-presence-of-multiple-sudorules-in-a-task.yml** file with the following content:

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

  tasks:
    # Ensure sudo command name: /usr/sbin/dmidecode is present
    - ipasudocmd:
        ipaadmin_password: "{{ ipaadmin_password }}"
        name: /usr/sbin/dmidecode

    # Ensure sudo command /usr/sbin/reboot is present
    - ipasudocmd:
        ipaadmin_password: "{{ ipaadmin_password }}"
        name: /usr/sbin/reboot

    # Ensure sudo command /usr/bin/yum is present
    - ipasudocmd:
        ipaadmin_password: "{{ ipaadmin_password }}"
        name: /usr/bin/yum

    # Ensure a sudo command group is present
    - ipasudocmdgroup:
        ipaadmin_password: "{{ ipaadmin_password }}"
        name: sudogroup01
        sudocmd:
          - /usr/sbin/dmidecode
          - /usr/sbin/reboot

    - name: Ensure multiple sudo rules are present using batch mode
      ipasudorule:
        ipaadmin_password: "{{ ipaadmin_password }}"
        sudorules:
          - name: testrule01
            user:
              - user01
```



```

- user02
group:
- usergroup01
allow_sudocmd:
- /usr/bin/yum
allow_sudocmdgroup:
- sudogroup01
- name: testrule02
hostgroup:
- hostgroup01
- hostgroup02

```

NOTE

Using the **sudorules** option, you can specify multiple **sudo** rule variables that only apply to a particular **sudo** rule. This **sudo** rule is defined by the **name** variable, which is the only mandatory variable for the **sudorules** option. In the example, the **user**, **group**, **allow_sudocmd**, and **allow_sudocmdgroup** variables are applied to the **testrule01 sudo** rule.

3. Save the file.
4. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory ensure-
presence-of-multiple-sudorules-in-a-task.yml
```

CHAPTER 21. ENSURING THE PRESENCE OF HOST-BASED ACCESS CONTROL RULES IN IDM USING ANSIBLE PLAYBOOKS

Ansible is an automation tool used to configure systems, deploy software, and perform rolling updates. It includes support for Identity Management (IdM).

Learn more about Identity Management (IdM) host-based access policies and how to define them using [Ansible](#).

21.1. 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.

21.2. ENSURING THE PRESENCE OF AN HBAC RULE IN IDM USING AN ANSIBLE PLAYBOOK

Follow this procedure to ensure the presence of a host-based access control (HBAC) rule in Identity Management (IdM) using an Ansible playbook.

Prerequisites

- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  # Ensure idm_user can access client.idm.example.com via the sshd service
  - ipahbacrule:
    ipaadmin_password: "{{ ipaadmin_password }}"
    name: login
    user: idm_user
    host: client.idm.example.com
    hbacsvc:
    - sshd
    state: present
```

3. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i
path_to_inventory_directory/inventory.file path_to_playbooks_directory/ensure-new-
hbacrule-present.yml
```

Verification

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

- See the **README-hbacsvc.md**, **README-hbacsvgroup.md**, and **README-hbacrule.md** files in the **/usr/share/doc/ansible-freeipa** directory.
- See the playbooks in the subdirectories of the **/usr/share/doc/ansible-freeipa/playbooks** directory.

CHAPTER 22. MANAGING IDM CERTIFICATES USING ANSIBLE

You can use the **ansible-freeipa ipacert** module to request, revoke, and retrieve SSL certificates for Identity Management (IdM) users, hosts and services. You can also restore a certificate that has been put on hold.

22.1. USING ANSIBLE TO REQUEST SSL CERTIFICATES FOR IDM HOSTS, SERVICES AND USERS

You can use the **ansible-freeipa ipacert** module to request SSL certificates for Identity Management (IdM) users, hosts and services. They can then use these certificates to authenticate to IdM.

Complete this procedure to request a certificate for an HTTP server from an IdM certificate authority (CA) using an Ansible playbook.

Prerequisites

- On the control node:
 - You are using Ansible version 2.14 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 in the **~/MyPlaybooks/** directory.
 - You have stored your **ipaadmin_password** in the **secret.yml** Ansible vault.
- Your IdM deployment has an integrated CA.

Procedure

1. Generate a certificate-signing request (CSR) for your user, host or service. For example, to use the **openssl** utility to generate a CSR for the **HTTP** service running on **client.idm.example.com**, enter:

```
# openssl req -new -newkey rsa:2048 -days 365 -nodes -keyout new.key -out new.csr -
subj '/CN=client.idm.example.com,O=IDM.EXAMPLE.COM'
```

As a result, the CSR is stored in **new.csr**.

2. Create your Ansible playbook file **request-certificate.yml** with the following content:

```
---
- name: Playbook to request a certificate
  hosts: ipaserver

  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml

  tasks:
    - name: Request a certificate for a web server
      ipacert:
        ipaadmin_password: "{{ ipaadmin_password }}"
        state: requested
```

```

csr: |
-----BEGIN CERTIFICATE REQUEST-----

MIGYMEwCAQAwGTEXMBUGA1UEAwwOZnJlZWlwYSBydWxlcyEwKjAFBgMrZXADIBs
Hlqlr4b/XNK+K8QLJKIzfVuNK0buBhLz3LAzY7QDEqAAMAUGAytIcANBAF4oSCbA
5aIPukCidnZJdr491G4LBE+URecYXsPknwYb+V+ONnf5ycZHyaFv+jkUBFGFeDgU
SYaXm/gF8cDYjQI=
-----END CERTIFICATE REQUEST-----
principal: HTTP/client.idm.example.com
register: cert

```

Replace the certificate request with the CSR from **new.csr**.

3. Request the certificate:

```

$ ansible-playbook --vault-password-file=password_file -v -i
<path_to_inventory_directory>/hosts <path_to_playbooks_directory>/request-
certificate.yml

```

Additional resources

- [The cert module in **ansible-freeipa** upstream docs](#)

22.2. USING ANSIBLE TO REVOKE SSL CERTIFICATES FOR IDM HOSTS, SERVICES AND USERS

You can use the **ansible-freeipa ipacert** module to revoke SSL certificates used by Identity Management (IdM) users, hosts and services to authenticate to IdM.

Complete this procedure to revoke a certificate for an HTTP server using an Ansible playbook. The reason for revoking the certificate is “keyCompromise”.

Prerequisites

- On the control node:
 - You are using Ansible version 2.14 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 in the **~/MyPlaybooks/** directory.
 - You have stored your **ipadmin_password** in the **secret.yml** Ansible vault.
 - You have obtained the serial number of the certificate, for example by entering the **openssl x509 -noout -text -in <path_to_certificate>** command. In this example, the serial number of the certificate is 123456789.
- Your IdM deployment has an integrated CA.

Procedure

1. Create your Ansible playbook file **revoke-certificate.yml** with the following content:

```

---
- name: Playbook to revoke a certificate
  hosts: ipaserver

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml

  tasks:
  - name: Revoke a certificate for a web server
    ipacert:
      ipadmin_password: "{{ ipadmin_password }}"
      serial_number: 123456789
      revocation_reason: "keyCompromise"
      state: revoked

```

2. Revoke the certificate:

```

$ ansible-playbook --vault-password-file=password_file -v -i
<path_to_inventory_directory>/hosts <path_to_playbooks_directory>/revoke-
certificate.yml

```

Additional resources

- [The cert module in **ansible-freeipa** upstream docs](#)
- [Reason Code](#) in RFC 5280

22.3. USING ANSIBLE TO RESTORE SSL CERTIFICATES FOR IDM USERS, HOSTS, AND SERVICES

You can use the **ansible-freeipa ipacert** module to restore a revoked SSL certificate previously used by an Identity Management (IdM) user, host or a service to authenticate to IdM.



NOTE

You can only restore a certificate that was put on hold. You may have put it on hold because, for example, you were not sure if the private key had been lost. However, now you have recovered the key and as you are certain that no-one has accessed it in the meantime, you want to reinstate the certificate.

Complete this procedure to use an Ansible playbook to release a certificate for a service enrolled into IdM from hold. This example describes how to release a certificate for an HTTP service from hold.

Prerequisites

- On the control node:
 - You are using Ansible version 2.14 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 in the `~/MyPlaybooks/` directory.

- You have stored your **ipaadmin_password** in the **secret.yml** Ansible vault.
- Your IdM deployment has an integrated CA.
- You have obtained the serial number of the certificate, for example by entering the **openssl x509 -noout -text -in path/to/certificate** command. In this example, the certificate serial number is 123456789.

Procedure

1. Create your Ansible playbook file **restore-certificate.yml** with the following content:

```
---
- name: Playbook to restore a certificate
  hosts: ipaserver
  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml

  tasks:
    - name: Restore a certificate for a web service
      ipacert:
        ipaadmin_password: "{{ ipaadmin_password }}"
        serial_number: 123456789
        state: released
```

2. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i
<path_to_inventory_directory>/hosts <path_to_playbooks_directory>/restore-
certificate.yml
```

Additional resources

- [The cert module in ansible-freeipa upstream docs](#)

22.4. USING ANSIBLE TO RETRIEVE SSL CERTIFICATES FOR IDM USERS, HOSTS, AND SERVICES

You can use the **ansible-freeipa ipacert** module to retrieve an SSL certificate issued for an Identity Management (IdM) user, host or a service, and store it in a file on the managed node.

Prerequisites

- On the control node:
 - You are using Ansible version 2.14 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 in the **~/MyPlaybooks/** directory.
 - You have stored your **ipaadmin_password** in the **secret.yml** Ansible vault.

- You have obtained the serial number of the certificate, for example by entering the **openssl x509 -noout -text -in <path_to_certificate>** command. In this example, the serial number of the certificate is 123456789, and the file in which you store the retrieved certificate is **cert.pem**.

Procedure

1. Create your Ansible playbook file **retrieve-certificate.yml** with the following content:

```
---
- name: Playbook to retrieve a certificate and store it locally on the managed node
  hosts: ipaserver

  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml

  tasks:
    - name: Retrieve a certificate and save it to file 'cert.pem'
      ipacert:
        ipadmin_password: "{{ ipadmin_password }}"
        serial_number: 123456789
        certificate_out: cert.pem
        state: retrieved
```

2. Retrieve the certificate:

```
$ ansible-playbook --vault-password-file=password_file -v -i
<path_to_inventory_directory>/hosts <path_to_playbooks_directory>/retrieve-
certificate.yml
```

Additional resources

- The **cert** module in [ansible-freeipa upstream docs](#)

CHAPTER 23. VAULTS IN IDM

Learn more about vaults in Identity Management (IdM).

23.1. VAULTS AND THEIR BENEFITS

You can use an Identity Management (IdM) vault to keep all your sensitive data stored securely but conveniently in one place.

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.

IdM vaults are characterized by the following:

- 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 all vaults.
- 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.

23.2. 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.

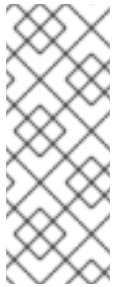
Each vault must have at least one owner. A vault can also have multiple owners.

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. In the context of IdM role-based access control (RBAC), a vault administrator is any IdM user with the **Vault Administrators** privilege.



NOTE

[Symmetric and asymmetric vaults](#) are protected with a password or key. Special access control rules apply for an administrator to:

- Access secrets in symmetric and asymmetric vaults.
- Change or reset the vault password or key.

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

- [Standard, symmetric and asymmetric vaults](#)
- [Using Ansible playbooks to manage role-based access control in IdM](#)

23.3. 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 secret inside a vault 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 asymmetric keys. Users archive the secret by using a public key and retrieve it by using a private key. Vault owners can both archive and retrieve secrets. Vault members can only archive secrets.

23.4. 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 23.1. 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.

23.5. VAULT CONTAINERS

A vault container is a collection of vaults. The [table below](#) lists the default vault containers that Identity Management (IdM) provides.

Table 23.2. 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
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.

23.6. BASIC IDM VAULT COMMANDS

You can use the basic commands outlined below 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 23.3. 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 user 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>

23.7. INSTALLING THE KEY RECOVERY AUTHORITY IN IDM

Follow this procedure to enable vaults in Identity Management (IdM) by installing the Key Recovery Authority (KRA) Certificate System (CS) component on a specific IdM server.

Prerequisites

- You are logged in as **root** on the IdM server.
- An IdM certificate authority is installed on the IdM server.
- You have the **Directory Manager** credentials.

Procedure

- Install the KRA:

```
# ipa-kra-install
```

**NOTE**

To make the vault service highly available and resilient, install the KRA on two IdM servers or more. Maintaining multiple KRA servers prevents data loss.



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.

Additional resources

- [Demoting or promoting hidden replicas](#)
- [The hidden replica mode](#)

CHAPTER 24. 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 a 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](#).

24.1. ENSURING THE PRESENCE OF A STANDARD USER VAULT IN IDM USING ANSIBLE

Follow this procedure to 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- You know the password of **idm_user**.

Procedure

1. Navigate to the **MyPlaybooks** directory:

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

2. Make a copy of the `/usr/share/doc/ansible-freeipa/playbooks/vault/ensure-standard-vault-is-present.yml` Ansible playbook file. For example:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/vault/ensure-standard-vault-is-present.yml ensure-standard-vault-is-present-copy.yml
```

3. Open the **ensure-standard-vault-is-present-copy.yml** file for editing.
4. 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 the modified Ansible playbook file for the current example:

```
---
- name: Tests
  hosts: ipaserver
  gather_facts: false

  tasks:
  - ipavault:
    ipaadmin_principal: idm_user
    ipaadmin_password: idm_user_password
    user: idm_user
    name: my_vault
    vault_type: standard
```

5. Save the file.
6. Run the playbook:

```
$ ansible-playbook -v -i inventory.file ensure-standard-vault-is-present-copy.yml
```

24.2. ARCHIVING A SECRET IN A STANDARD USER VAULT IN IDM USING ANSIBLE

Follow this procedure to 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the **~/MyPlaybooks/** directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.

- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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 **MyPlaybooks** directory:

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

2. Make a copy of the **/usr/share/doc/ansible-freeipa/playbooks/vault/data-archive-in-symmetric-vault.yml** Ansible playbook file. For example:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/vault/data-archive-in-symmetric-vault.yml data-archive-in-symmetric-vault-copy.yml
```

3. Open the **data-archive-in-standard-vault-copy.yml** file for editing.
4. 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 the modified Ansible playbook file for the current example:

```
---
- name: Tests
  hosts: ipaserver
  gather_facts: false

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  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
```

5. Save the file.

6. Run the playbook:

```
$ ansible-playbook -v -i inventory.file data-archive-in-standard-vault-copy.yml
```

24.3. RETRIEVING A SECRET FROM A STANDARD USER VAULT IN IDM USING ANSIBLE

Follow this procedure to 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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 **MyPlaybooks** directory:

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

2. Make a copy of the `/usr/share/doc/ansible-freeipa/playbooks/vault/data-archive-in-symmetric-vault.yml` Ansible playbook file. For example:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/vault/retrieve-data-symmetric-vault.yml  
retrieve-data-symmetric-vault-copy.yml
```

3. 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
```

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 is the modified Ansible playbook file for the current example:

```
---
- name: Tests
  hosts: ipahost
  gather_facts: false

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  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

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.

Additional resources

- 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 25. USING ANSIBLE TO MANAGE IDM SERVICE VAULTS: STORING AND RETRIEVING SECRETS

This section describes 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 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

- All the IdM servers in the **ipaserver** host category defined in the playbooks below have the Key Recovery Authority (KRA) Certificate System component installed on them. For details, see [Installing the Key Recovery Authority in IdM](#).

In the procedures:

- The IdM **admin** user 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**.

25.1. ENSURING THE PRESENCE OF AN ASYMMETRIC SERVICE VAULT IN IDM USING ANSIBLE

Follow this procedure to 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 to retrieve the secret in the vault. The vault members will be able to retrieve the file from any IdM client.

Prerequisites

- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the `secret.yml` Ansible vault stores your `ipadmin_password`.
- The target node, that is the node on which the `ansible-freeipa` module is executed, is part of the IdM domain as an IdM client, server or replica.

Procedure

1. Navigate to the `MyPlaybooks` directory:

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

2. Make a copy of the `/usr/share/doc/ansible-freeipa/playbooks/vault/ensure-asymmetric-vault-is-present.yml` Ansible playbook file. For example:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/vault/ensure-asymmetric-vault-is-present.yml ensure-asymmetric-vault-is-present-copy.yml
```

3. 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
```

4. Open the `ensure-asymmetric-vault-is-present-copy.yml` file for editing.
5. Add a task that copies the `service-public.pem` public key from the Ansible controller to the `server.idm.example.com` server.
6. Modify the rest of the file by setting the following variables in the `ipavault` task section:
 - Set the `ipadmin_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
  gather_facts: false
  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  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:
        ipadmin_password: "{{ ipadmin_password }}"
        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
```

7. Save the file.

8. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory.file ensure-
asymmetric-service-vault-is-present-copy.yml
```

25.2. ADDING MEMBER SERVICES TO AN ASYMMETRIC VAULT USING ANSIBLE

Follow this procedure to 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the **~/MyPlaybooks/** directory, you have created an **Ansible inventory file** with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.

- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- You have [created an asymmetric vault](#) to store the service secret.

Procedure

1. Navigate to the **MyPlaybooks** directory:

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

2. Make a copy of the `/usr/share/doc/ansible-freeipa/playbooks/vault/data-archive-in-asymmetric-vault.yml` Ansible playbook file. For example:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/vault/data-archive-in-asymmetric-vault.yml data-archive-in-asymmetric-vault-copy.yml
```

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/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
  gather_facts: false

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - ipavault:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: secret_vault
      service: HTTP/webserver1.idm.example.com
      services:
      - HTTP/webserver2.idm.example.com
      - HTTP/webserver3.idm.example.com
      action: member
```

5. Save the file.
6. Run the playbook:


```
$ ansible-playbook --vault-password-file=password_file -v -i inventory.file add-
services-to-an-asymmetric-vault.yml
```

25.3. STORING AN IDM SERVICE SECRET IN AN ASYMMETRIC VAULT USING ANSIBLE

Follow this procedure to 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 to retrieve the secret from the vault. The service will be able to retrieve the file from any IdM client.

Prerequisites

- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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 **MyPlaybooks** directory:

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

2. Make a copy of the `/usr/share/doc/ansible-freeipa/playbooks/vault/data-archive-in-asymmetric-vault.yml` Ansible playbook file. For example:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/vault/data-archive-in-asymmetric-
vault.yml data-archive-in-asymmetric-vault-copy.yml
```

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 **ipadmin_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
  gather_facts: false

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - ipavault:
      ipadmin_password: "{{ ipadmin_password }}"
      name: secret_vault
      service: HTTP/webserver1.idm.example.com
      in: "{{ lookup('file', 'private-key-to-an-externally-signed-certificate.pem') | b64encode }}"
      action: member
```

5. Save the file.
6. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory.file data-
archive-in-asymmetric-vault-copy.yml
```

25.4. RETRIEVING A SERVICE SECRET FOR AN IDM SERVICE USING ANSIBLE

Follow this procedure to 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.

- The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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 **MyPlaybooks** directory:

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

2. Make a copy of the **/usr/share/doc/ansible-freeipa/playbooks/vault/retrieve-data-asymmetric-vault.yml** Ansible playbook file. For example:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/vault/retrieve-data-asymmetric-vault.yml  
retrieve-data-asymmetric-vault-copy.yml
```

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. Open the **retrieve-data-asymmetric-vault-copy.yml** file for editing.
5. 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Retrieve data from the service vault
    ipavault:
      ipaadmin_password: "{{ ipaadmin_password }}"
      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
```

6. 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: true
      mode: 0600
```

7. 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
```

8. Save the file.
9. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory.file retrieve-
data-asymmetric-vault-copy.yml
```

25.5. CHANGING AN IDM SERVICE VAULT SECRET WHEN COMPROMISED USING ANSIBLE

Follow this procedure to 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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:

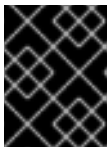
```
$ cd /usr/share/doc/ansible-freeipa/playbooks/vault
```

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. 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
```

4. Open the **data-archive-in-asymmetric-vault-copy.yml** file for editing.

5. 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
  gather_facts: false

  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
    - ipavault:
        ipaadmin_password: "{{ ipaadmin_password }}"
        name: secret_vault
```

```

    service: HTTP/webserver.idm.example.com
    in: "{{ lookup('file', 'new-private-key-to-an-externally-signed-certificate.pem') | b64encode
  }}"
  action: member

```

6. Save the file.

7. Run the playbook:

```

$ ansible-playbook --vault-password-file=password_file -v -i inventory.file data-
archive-in-asymmetric-vault-copy.yml

```

8. Open the `retrieve-data-asymmetric-vault-copy.yml` file for editing.

9. 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Retrieve data from the service vault
    ipavault:
      ipaadmin_password: "{{ ipaadmin_password }}"
      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

```

10. 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: true
  gather_facts: false
  tasks:
[...]
```

```

- name: Retrieve data file
  fetch:
    src: new-private-key-to-an-externally-signed-certificate.pem
    dest: ./
    flat: true
    mode: 0600
```

11. 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 webserver listed in the **webserver** section of the inventory file:

```

---
- name: Send data file to webserver
  become: true
  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
```

12. Save the file.
13. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory.file retrieve-data-asymmetric-vault-copy.yml
```

25.6. ADDITIONAL RESOURCES

- See the README-vault.md Markdown file in the **/usr/share/doc/ansible-freeipa/** directory.
- See the sample playbooks in the **/usr/share/doc/ansible-freeipa/playbooks/vault/** directory.

CHAPTER 26. 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.](#)

26.1. ENSURING THE PRESENCE OF AN HTTP SERVICE IN IDM USING AN ANSIBLE PLAYBOOK

Follow this procedure 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
  gather_facts: false

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  # Ensure service is present
  - ipaservice:
    ipaadmin_password: "{{ ipaadmin_password }}"
    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. Specify the playbook file, the file storing the password protecting the `secret.yml` file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i
   path_to_inventory_directory/inventory.file /usr/share/doc/ansible-
   freeipa/playbooks/service/service-is-present-copy.yml
```

Verification

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

- To secure the communication between the HTTP server and browser clients, see [adding TLS encryption to an Apache HTTP Server](#).

- To request a certificate for the HTTP service, see the procedure described in [Obtaining an IdM certificate for a service using certmonger](#).

26.2. ENSURING THE PRESENCE OF MULTIPLE SERVICES IN IDM ON AN IDM CLIENT USING A SINGLE ANSIBLE TASK

You can use the **ansible-freeipa ipaservice** module to add, modify, and delete multiple Identity Management (IdM) services with a single Ansible task. For that, use the **services** option of the **ipaservice** module.

Using the **services** option, you can also specify multiple service variables that only apply to a particular service. Define this service by the **name** variable, which is the only mandatory variable for the **services** option.

Complete this procedure to ensure the presence of the **HTTP/client01.idm.example.com@IDM.EXAMPLE.COM** and the **ftp/client02.idm.example.com@IDM.EXAMPLE.COM** services in IdM with a single task.

Prerequisites

- On the control node:
 - You are using Ansible version 2.14 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 in the **~/MyPlaybooks/** directory.
 - You are using RHEL 9.3 and later.
 - You have stored your **ipaadmin_password** in the **secret.yml** Ansible vault.

Procedure

1. Create your Ansible playbook file **add-http-and-ftp-services.yml** with the following content:

```
---
- name: Playbook to add multiple services in a single task
  hosts: ipaserver
  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml

  tasks:
    - name: Add HTTP and ftp services
      ipaservice:
        ipaadmin_password: "{{ ipaadmin_password }}"
        services:
          - name: HTTP/client01.idm.example.com@IDM.EXAMPLE.COM
          - name: ftp/client02.idm.example.com@IDM.EXAMPLE.COM
```

2. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory add-http-and-ftp-services.yml
```

Additional resources

- The service module in [ansible-freeipa upstream docs](#)

26.3. ENSURING THE PRESENCE OF AN HTTP SERVICE IN IDM ON A NON-IDM CLIENT USING AN ANSIBLE PLAYBOOK

Follow this procedure 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 [enrolled the HTTP service into IdM](#).
- You have the IdM administrator password.
- The DNS A record – or the AAAA record if IPv6 is used – for the host is available.
- If the server runs RHEL 9.2 or later and the FIPS mode is enabled, clients must either support the Extended Master Secret (EMS) extension or use TLS 1.3. TLS 1.2 connections without EMS fail. For more information, see the Red Hat Knowledgebase solution [TLS extension "Extended Master Secret" enforced](#).

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 **ipadmin_password** and **name** variables in the **ipaservice** task:

```
---
- name: Playbook to manage IPA service.
  hosts: ipaserver
  gather_facts: false
```

```
vars_files:
- /home/user_name/MyPlaybooks/secret.yml
tasks:
# Ensure service is present
- ipaservice:
    ipaadmin_password: "{{ ipaadmin_password }}"
    name: HTTP/www2.example.com
    skip_host_check: true
```

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. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i
path_to_inventory_directory/inventory.file /usr/share/doc/ansible-
freeipa/playbooks/service/service-is-present-without-host-check-copy.yml
```

Verification

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

- To secure the communication, see [adding TLS encryption to an Apache HTTP Server](#) .

26.4. ENSURING THE PRESENCE OF AN HTTP SERVICE ON AN IDM CLIENT WITHOUT DNS USING AN ANSIBLE PLAYBOOK

Follow this procedure 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
  gather_facts: false

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  # Ensure service is present
  - ipaservice:
    ipaadmin_password: "{{ ipaadmin_password }}"
    name: HTTP/ihavenodns.info
    force: true
```

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. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i
path_to_inventory_directory/inventory.file /usr/share/doc/ansible-freeipa/playbooks/service/service-is-present-with-host-force-copy.yml
```

Verification

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

- To secure the communication, see [adding TLS encryption to an Apache HTTP Server](#) .

26.5. ENSURING THE PRESENCE OF AN EXTERNALLY SIGNED CERTIFICATE IN AN IDM SERVICE ENTRY USING AN ANSIBLE PLAYBOOK

Follow this procedure 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 (CLI):

```
$ openssl x509 -outform der -in cert1.pem -out cert1.der
```

- 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...
```

- Copy the certificate from the standard output to the clipboard.
- 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
  gather_facts: false

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  # Ensure service certificate is present
  - ipaservice:
    ipaadmin_password: "{{ ipaadmin_password }}"
    name: HTTP/client.idm.example.com
    certificate: |
      - MIICBjCCA8CFHnm32VcXaUDGfEGdDL/...
      [...]
    action: member
    state: present
```

- 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 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.
- Save and exit the file.
- Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i
path_to_inventory_directory/inventory.file /usr/share/doc/ansible-
freeipa/playbooks/service/service-member-certificate-present-copy.yml
```

Verification

- 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.

26.6. 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.

Follow this procedure to 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the **~/MyPlaybooks/** directory, you have created an **Ansible inventory file** with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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
```

- 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
```

- 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
```

- Open the **/usr/share/doc/ansible-freeipa/playbooks/service/service-member-allow_create_keytab-present-copy.yml** Ansible playbook file for editing.
- 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Service HTTP/client.idm.example.com members allow_create_keytab present for
    user01
    ipaservice:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: HTTP/client.idm.example.com
      allow_create_keytab_user:
      - user01
      action: member
```

- Save the file.

7. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i
path_to_inventory_directory/inventory.file /usr/share/doc/ansible-
freeipa/playbooks/service/service-member-allow_create_keytab-present-copy.yml
```

Verification

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**.

26.7. 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.

Follow this procedure to 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.

- The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
- The example assumes that the `secret.yml` Ansible vault stores your `ipaadmin_password`.
- The target node, that is the node on which the `ansible-freeipa` module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Service HTTP/client.idm.example.com members allow_retrieve_keytab present for
    user01
    ipaservice:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: HTTP/client.idm.example.com
      allow_retrieve_keytab_user:
        - user01
      action: member

```

6. Save the file.
7. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```

$ ansible-playbook --vault-password-file=password_file -v -i
path_to_inventory_directory/inventory.file /usr/share/doc/ansible-
freeipa/playbooks/service/service-member-allow_retrieve_keytab-present-copy.yml

```

Verification

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**.

26.8. 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.

Follow this procedure to create the principal alias of **HTTP/mycompany.idm.example.com** for the HTTP service running on **client.idm.example.com**.

Prerequisites

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Service HTTP/client.idm.example.com member principals
    host/mycompany.idm.example.com present
    ipaservice:
      ipadmin_password: "{{ ipadmin_password }}"
      name: HTTP/client.idm.example.com
      principal:
        - host/mycompany.idm.example.com
      action: member
```

6. Save the file.
7. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -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

- [Managing Kerberos principal aliases for users, hosts, and services](#)

26.9. ENSURING THE ABSENCE OF AN HTTP SERVICE IN IDM USING AN ANSIBLE PLAYBOOK

Follow this procedure 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
  gather_facts: false

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  # Ensure service is absent
  - ipaservice:
    ipaadmin_password: "{{ ipaadmin_password }}"
    name: HTTP/client.idm.example.com
    state: absent
```

6. Save and exit the file.
7. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i
   path_to_inventory_directory/inventory.file /usr/share/doc/ansible-
   freeipa/playbooks/service/service-is-absent-copy.yml
```

Verification

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.

26.10. ADDITIONAL RESOURCES

- See the **README-service.md** Markdown file in the **/usr/share/doc/ansible-freeipa/** directory.
- See sample playbooks in the **/usr/share/doc/ansible-freeipa/playbooks/config** directory.

CHAPTER 27. 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](#)
- [The **action: member** option in ipadnsconfig ansible-freeipa modules](#)
- [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](#)

27.1. 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:

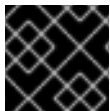
1. The DNS installation script creates an `/etc/NetworkManager/conf.d/zzz-ipa.conf` **NetworkManager** configuration file to control the search order and DNS server list:

```
# 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.

27.2. ENSURING THE PRESENCE OF A DNS GLOBAL FORWARDER IN IDM USING ANSIBLE

Follow this procedure to 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the `secret.yml` Ansible vault stores your `ipadmin_password`.

- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  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 --vault-password-file=password_file -v -i inventory.file ensure-presence-
of-a-global-forwarder.yml
```

Additional resources

- The **README-dnsconfig.md** file in the `/usr/share/doc/ansible-freeipa/` directory

27.3. ENSURING THE ABSENCE OF A DNS GLOBAL FORWARDER IN IDM USING ANSIBLE

Follow this procedure to 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 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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:

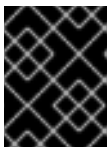
```
$ cp forwarders-absent.yml ensure-absence-of-a-global-forwarder.yml
```

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. Set the **action** variable to **member**.
 - e. Verify the **state** is set to **absent**.

This the modified Ansible playbook file for the current example:

```
---
- name: Playbook to ensure the absence of a global forwarder in IdM DNS
  hosts: ipaserver

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  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
      action: member
      state: absent
```



IMPORTANT

If you only use the **state: absent** option in your playbook without also using **action: member**, the playbook fails.

6. Save the file.
7. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory.file ensure-absence-of-a-global-forwarder.yml
```

Additional resources

- The **README-dnsconfig.md** file in the `/usr/share/doc/ansible-freeipa/` directory
- The **action: member** option in `ipadnsconfig` `ansible-freeipa` modules

27.4. THE ACTION: MEMBER OPTION IN IPADNSCONFIG ANSIBLE-FREEIPA MODULES

Excluding global forwarders in Identity Management (IdM) by using the **ansible-freeipa** `ipadnsconfig` module requires using the **action: member** option in addition to the **state: absent** option. If you only use **state: absent** in your playbook without also using **action: member**, the playbook fails. Consequently, to remove all global forwarders, you must specify all of them individually in the playbook. In contrast, the **state: present** option does not require **action: member**.

The following table provides configuration examples for both adding and removing DNS global forwarders that demonstrate the correct use of the `action: member` option. The table shows, in each line:

- The global forwarders configured before executing a playbook
- An excerpt from the playbook
- The global forwarders configured after executing the playbook

Table 27.1. `ipadnsconfig` management of global forwarders

Forwarders before	Playbook excerpt	Forwarders after
8.8.6.6	<pre>[...] tasks: - name: Ensure the presence of DNS global forwarder 8.8.6.7 ipadnsconfig: forwarders: - ip_address: 8.8.6.7 state: present</pre>	8.8.6.7
8.8.6.6	<pre>[...] tasks: - name: Ensure the presence of DNS global forwarder 8.8.6.7 ipadnsconfig: forwarders: - ip_address: 8.8.6.7 action: member state: present</pre>	8.8.6.6, 8.8.6.7

Forwarders before	Playbook excerpt	Forwarders after
8.8.6.6, 8.8.6.7	<pre>[...] tasks: - name: Ensure the absence of DNS global forwarder 8.8.6.7 ipadnsconfig: forwarders: - ip_address: 8.8.6.7 state: absent</pre>	Trying to execute the playbook results in an error. The original configuration - 8.8.6.6, 8.8.6.7 - is left unchanged.
8.8.6.6, 8.8.6.7	<pre>[...] tasks: - name: Ensure the absence of DNS global forwarder 8.8.6.7 ipadnsconfig: forwarders: - ip_address: 8.8.6.7 action: member state: absent</pre>	8.8.6.6

27.5. 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)

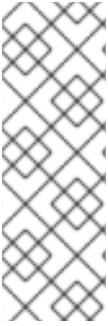
The IdM BIND service forwards DNS queries to the configured forwarder. If a query fails because of a server error or timeout, BIND falls back to the recursive resolution using servers on the Internet. The **forward first** policy is the default policy, and it is suitable for optimizing DNS traffic.

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.

27.6. USING AN ANSIBLE PLAYBOOK TO ENSURE THAT THE FORWARD FIRST POLICY IS SET IN IDM DNS GLOBAL CONFIGURATION

Follow this procedure to 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the **~/MyPlaybooks/** directory, you have created an **Ansible inventory file** with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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: "{{ ipaadmin_password }}"
        forward_policy: first
```

6. Save the file.
7. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory.file set-forward-policy-to-first.yml
```

Additional resources

- [DNS forward policies in IdM](#)
- The **README-dnsconfig.md** file in the **/usr/share/doc/ansible-freeipa/** directory
- For more sample playbooks, see the **/usr/share/doc/ansible-freeipa/playbooks/dnsconfig** directory.

27.7. USING AN ANSIBLE PLAYBOOK TO ENSURE THAT GLOBAL FORWARDERS ARE DISABLED IN IDM DNS

Follow this procedure to 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 option is the IdM equivalent of specifying an empty list of forwarders in BIND configuration.

Prerequisites

- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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 **ipadmin_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: "{{ ipaadmin_password }}"
      forward_policy: none

```

6. Save the file.

7. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory.file disable-global-forwarders-copy.yml
```

Additional resources

- [DNS forward policies in IdM](#)
- The **README-dnsconfig.md** file in the `/usr/share/doc/ansible-freeipa/` directory
- Sample playbooks in the `/usr/share/doc/ansible-freeipa/playbooks/dnsconfig` directory

27.8. USING AN ANSIBLE PLAYBOOK TO ENSURE THAT SYNCHRONIZATION OF FORWARD AND REVERSE LOOKUP ZONES IS DISABLED IN IDM DNS

Follow this procedure to use an Ansible playbook to ensure that forward and reverse lookup zones are not synchronized in IdM DNS.

Prerequisites

- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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:
        ipaadmin_password: "{{ ipaadmin_password }}"
        allow_sync_ptr: no
```

6. Save the file.
7. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory.file disallow-reverse-sync-copy.yml
```

Additional resources

- The `README-dnsconfig.md` file in the `/usr/share/doc/ansible-freeipa/` directory
- For more sample playbooks, see the `/usr/share/doc/ansible-freeipa/playbooks/dnsconfig` directory.

CHAPTER 28. 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.

- [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 Ansible to install an IdM server with integrated DNS, see [Installing an Identity Management server using an Ansible playbook](#).

28.1. SUPPORTED DNS ZONE TYPES

This section describes the two types of DNS zones that Identity Management (IdM) supports, *primary* and *forward* zones, and includes an example scenario for DNS forwarding.



NOTE

This section 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 by 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, also known as forwarders, may or may not be authoritative for the query.

Example 28.1. 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 by 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 IdM DNS server forwards the query to an AD domain controller specified in the **ad.example.com** IdM DNS forward zone.

28.2. 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 on the command line (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 [Using Ansible playbooks to manage IdM DNS zones](#).

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 28.1. IdM DNS zone attributes

Attribute	ansible-freeipa variable	Description
-----------	--------------------------	-------------

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 example, 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>

Attribute	ansible-freeipa variable	Description
Allow transfer	allow_transfer =string	Gives a list of IP addresses or network names which are allowed to transfer the given zone, separated by semicolons (;). Zone transfers are disabled by default. The default allow_transfer value is none .
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 =IP_address	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

- See the **README-dnszone.md** file in the `/usr/share/doc/ansible-freeipa/` directory.

28.3. USING ANSIBLE TO CREATE A PRIMARY ZONE IN IDM DNS

Follow this procedure to use an Ansible playbook to ensure that a primary DNS zone exists. In the example used in the procedure below, you ensure the presence of the **zone.idm.example.com** DNS zone.

Prerequisites

- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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-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: "{{ ipaadmin_password }}"
        zone_name: zone.idm.example.com
        state: present
```

6. Save the file.
7. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory.file dnszone-present-copy.yml
```

Additional resources

- [Supported DNS zone types](#)
- The **README-dnszone.md** file in the **/usr/share/doc/ansible-freeipa/** directory
- Sample Ansible playbooks in the **/usr/share/doc/ansible-freeipa/playbooks/dnszone** directory

28.4. USING AN ANSIBLE PLAYBOOK TO ENSURE THE PRESENCE OF A PRIMARY DNS ZONE IN IDM WITH MULTIPLE VARIABLES

Follow this procedure to 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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 **ipadmin_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.

**WARNING**

DNSSEC is only available as Technology Preview in IdM.

- 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:
        ipaadmin_password: "{{ ipaadmin_password }}"
        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 --vault-password-file=password_file -v -i inventory.file dnszone-all-params-copy.yml
```

Additional resources

- See [Supported DNS zone types](#).
- See [Configuration attributes of primary IdM DNS zones](#).
- See the **README-dnszone.md** file in the `/usr/share/doc/ansible-freeipa/` directory.
- See sample Ansible playbooks in the `/usr/share/doc/ansible-freeipa/playbooks/dnszone` directory.

28.5. USING AN ANSIBLE PLAYBOOK TO ENSURE THE PRESENCE OF A ZONE FOR REVERSE DNS LOOKUP WHEN AN IP ADDRESS IS GIVEN

Follow this procedure to 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 configured your Ansible control node to meet the following requirements:

- You are using Ansible version 2.14 or later.
- You have installed the **ansible-freeipa** package.
- The example assumes that in the `~/MyPlaybooks/` directory, you have created an **Ansible inventory file** with the fully-qualified domain name (FQDN) of the IdM server.
- The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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:
        ipaadmin_password: "{{ ipaadmin_password }}"
        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 --vault-password-file=password_file -v -i inventory.file dnszone-  
reverse-from-ip-copy.yml
```

Additional resources

- [Supported DNS zone types](#)
- The **README-dnszone.md** file in the `/usr/share/doc/ansible-freeipa/` directory.
- Sample Ansible playbooks in the `/usr/share/doc/ansible-freeipa/playbooks/dnszone` directory

CHAPTER 29. 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.

- [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](#)

29.1. 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 29.1. 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 29.2. 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.

29.2. 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.

29.3. 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

- [Configuration attributes of primary IdM DNS zones](#)

29.4. 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

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the `secret.yml` Ansible vault stores your `ipadmin_password`.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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 **ipadmin_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

  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
    - name: Ensure that the "germany" location is present
```

```

ipalocation:
  ipaadmin_password: "{{ ipaadmin_password }}"
  name: germany

```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```

$ ansible-playbook --vault-password-file=password_file -v -i inventory location-
present-copy.yml

```

Additional resources

- [Assigning an IdM server to a DNS location using the IdM Web UI](#)
- [Assigning an IdM server to a DNS location using the IdM CLI](#)

29.5. 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure that the "germany" location is absent
    ipalocation:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: germany
      state: absent
```

5. Save the file.
6. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory location-absent-copy.yml
```

29.6. ADDITIONAL RESOURCES

- See the **README-location.md** file in the **/usr/share/doc/ansible-freeipa/** directory.
- See sample Ansible playbooks in the **/usr/share/doc/ansible-freeipa/playbooks/location** directory.

CHAPTER 30. MANAGING DNS FORWARDING IN IDM

Follow these procedures to configure DNS global forwarders and DNS forward zones in the Identity Management (IdM) Web UI, the IdM CLI, and using Ansible:

- [The two roles of an IdM DNS server](#)
- [DNS forward policies in IdM](#)
- [Adding a global forwarder in the IdM Web UI](#)
- [Adding a global forwarder in the CLI](#)
- [Adding a DNS Forward Zone in the IdM Web UI](#)
- [Adding a DNS Forward Zone in the CLI](#)
- [Establishing a DNS Global Forwarder in IdM using Ansible](#)
- [Ensuring the presence of a DNS global forwarder in IdM using Ansible](#)
- [Ensuring the absence of a DNS global forwarder in IdM using Ansible](#)
- [Ensuring DNS Global Forwarders are disabled in IdM using Ansible](#)
- [Ensuring the presence of a DNS Forward Zone in IdM using Ansible](#)
- [Ensuring a DNS Forward Zone has multiple forwarders in IdM using Ansible](#)
- [Ensuring a DNS Forward Zone is disabled in IdM using Ansible](#)
- [Ensuring the absence of a DNS Forward Zone in IdM using Ansible](#)

30.1. 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.

30.2. 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 30.1. 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.

30.3. ADDING A GLOBAL FORWARDER IN THE IDM WEB UI

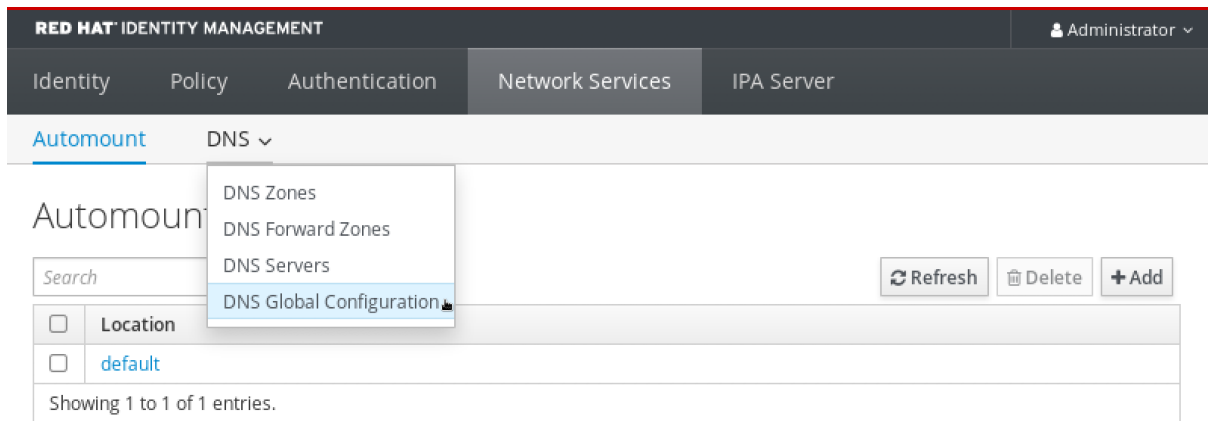
Follow this procedure to add a global DNS forwarder in the Identity Management (IdM) Web UI.

Prerequisites

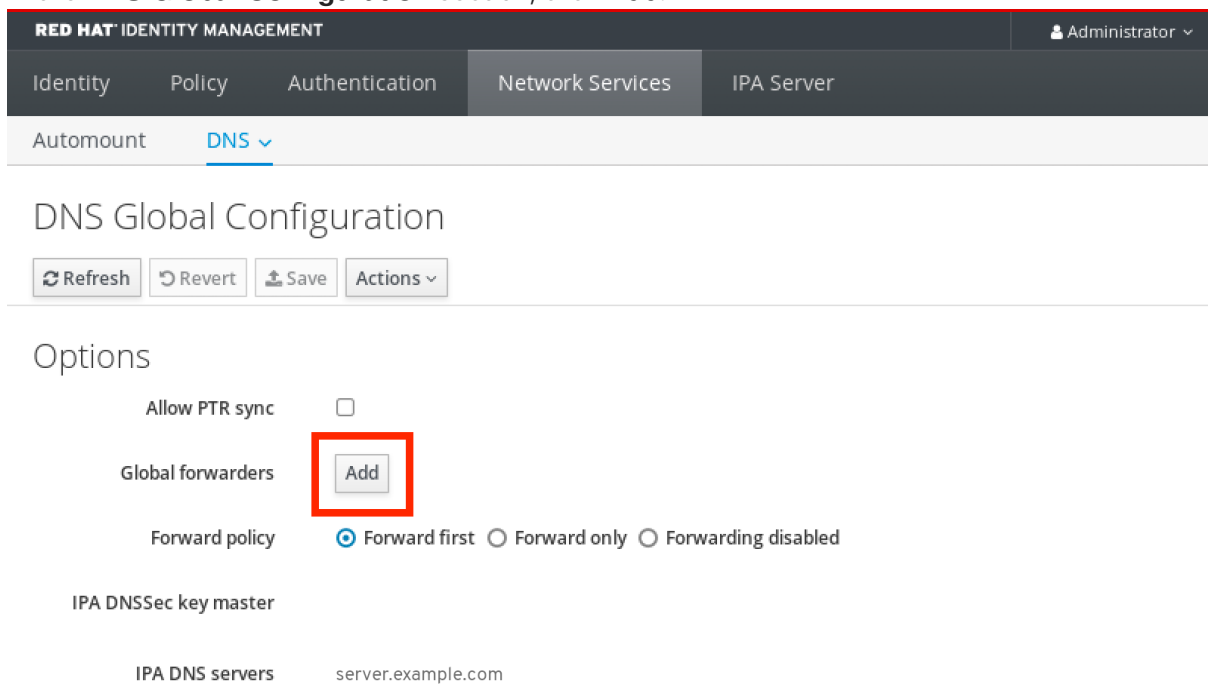
- You are logged in to the IdM WebUI as IdM administrator.
- You know the Internet Protocol (IP) address of the DNS server to forward queries to.

Procedure

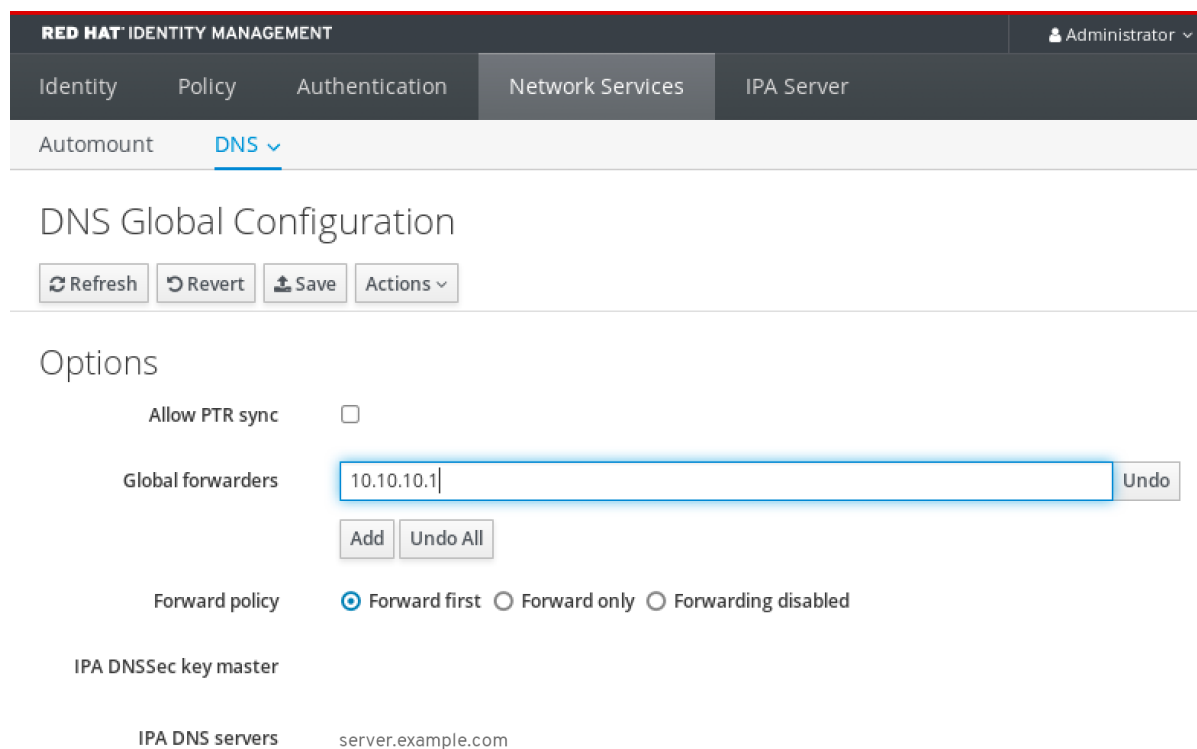
1. In the IdM Web UI, select **Network Services** → **DNS Global Configuration** → **DNS**.



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](#)

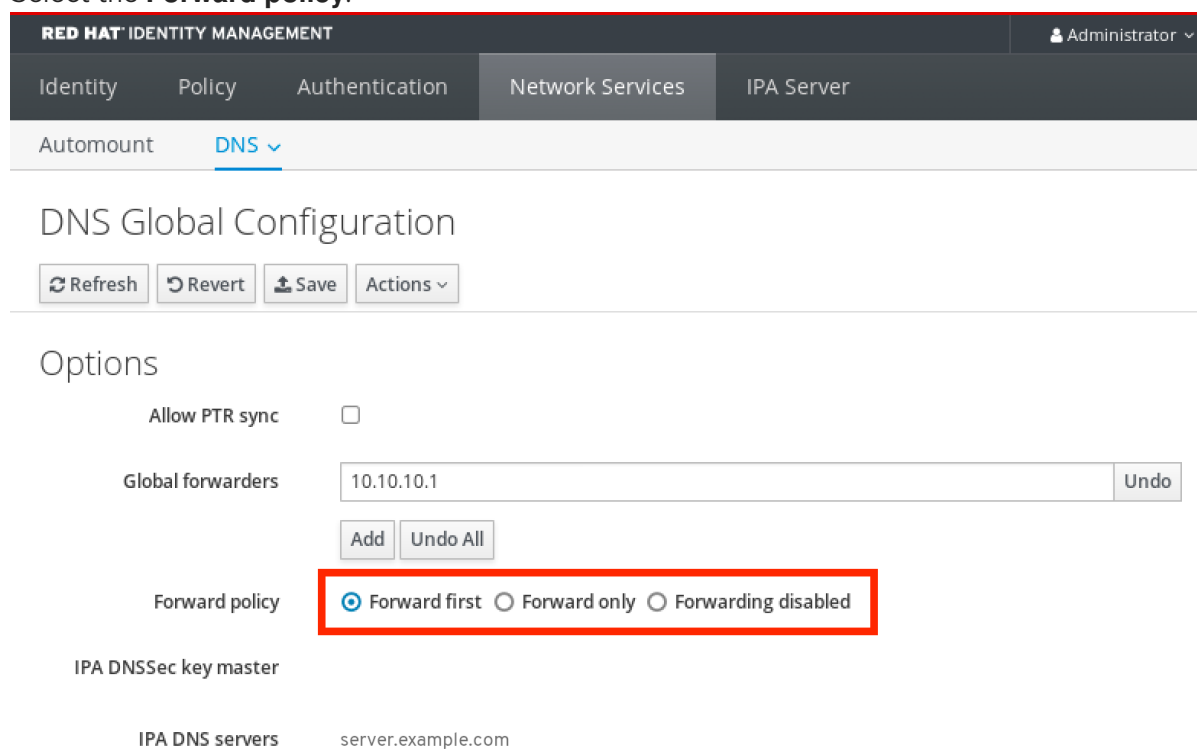
[Add](#) [Undo All](#)

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

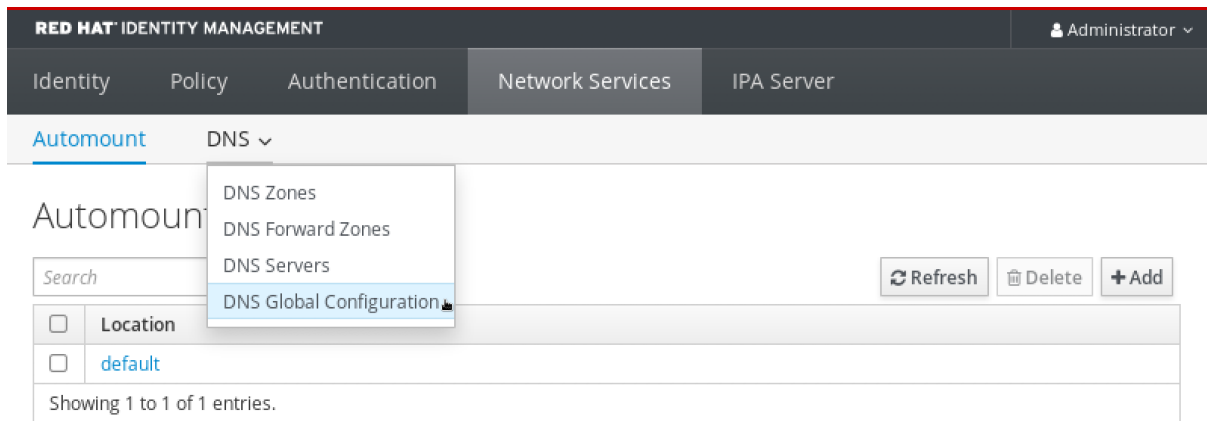
IPA DNSSec key master

IPA DNS servers server.example.com

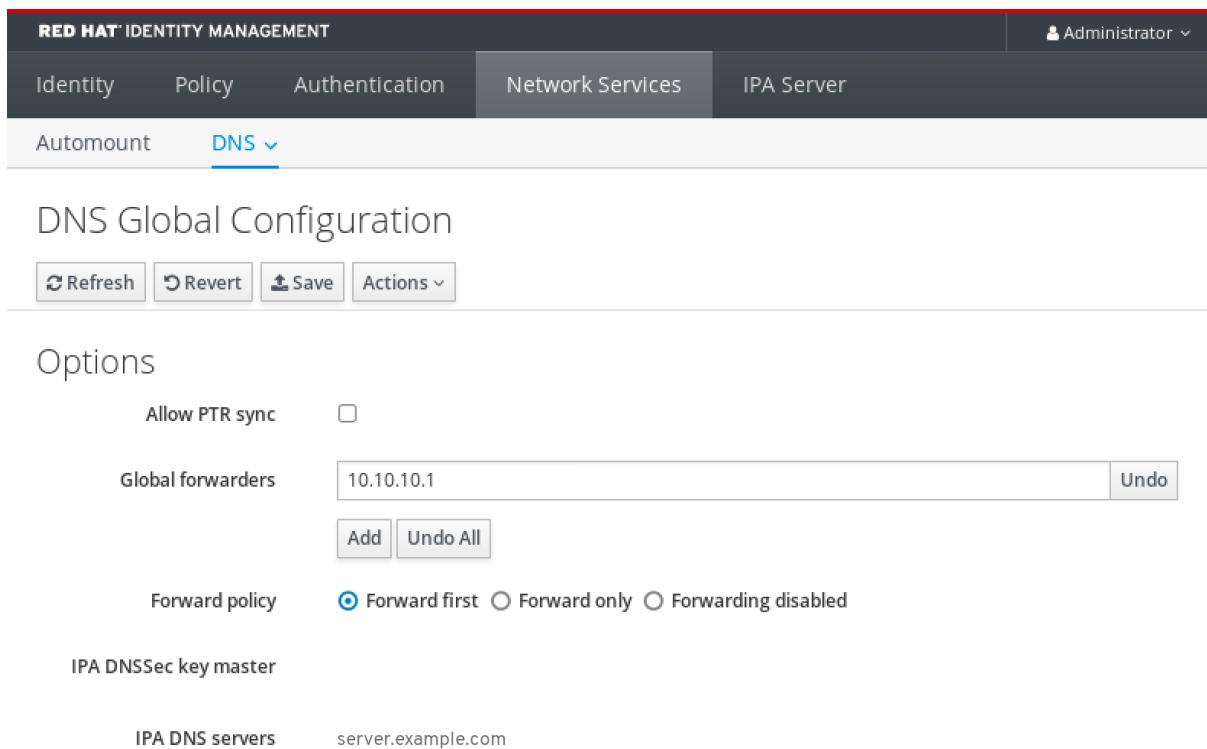
5. Click **Save** at the top of the window.

Verification

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.



30.4. ADDING A GLOBAL FORWARDER IN THE CLI

Follow this procedure to add a global DNS forwarder by using the command line (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

- 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
```

30.5. ADDING A DNS FORWARD ZONE IN THE IDM WEB UI

Follow this procedure 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.

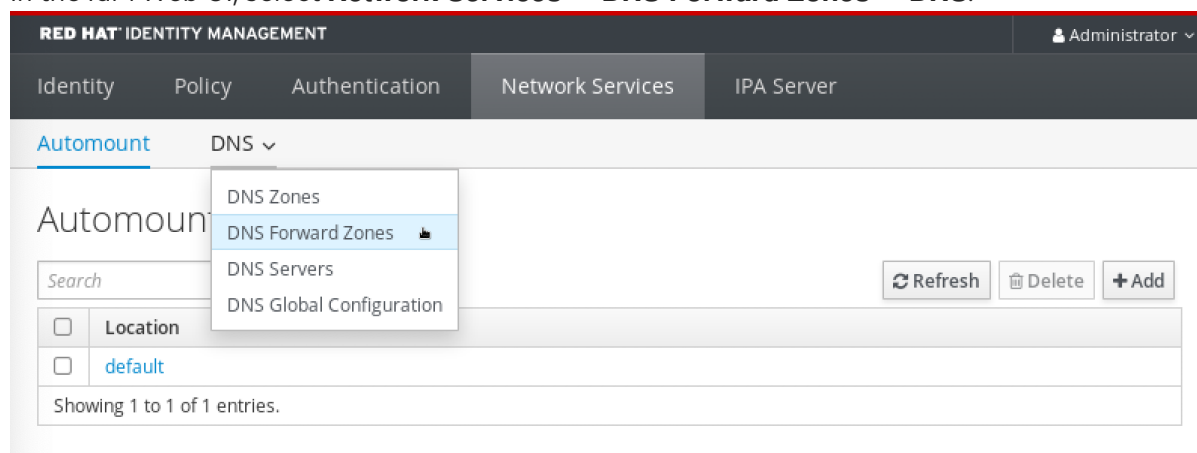
When creating a new DNS zone, Red Hat recommends to always use standard DNS delegation using nameserver (NS) records and to avoid forward zones. In most cases, using a global forwarder is sufficient, and forward zones are not necessary.

Prerequisites

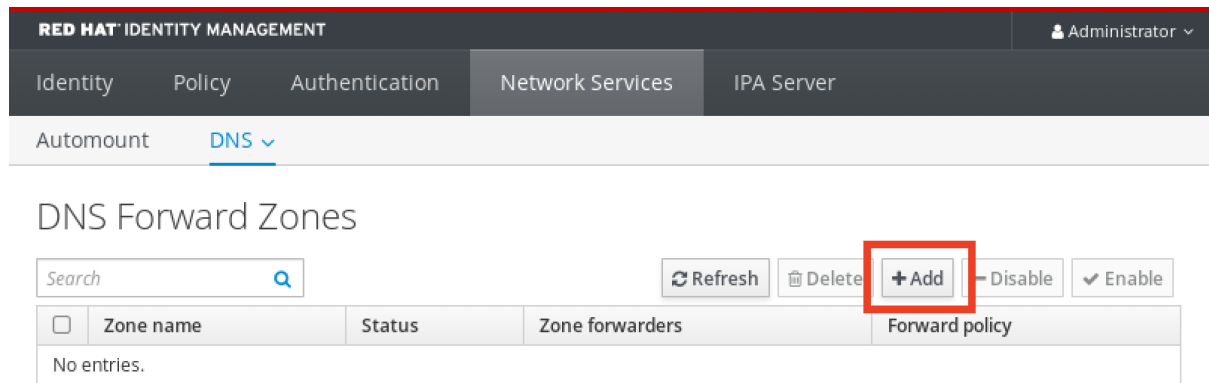
- You are logged in to the IdM WebUI as IdM administrator.
- You know the Internet Protocol (IP) address of the DNS server to forward queries to.

Procedure

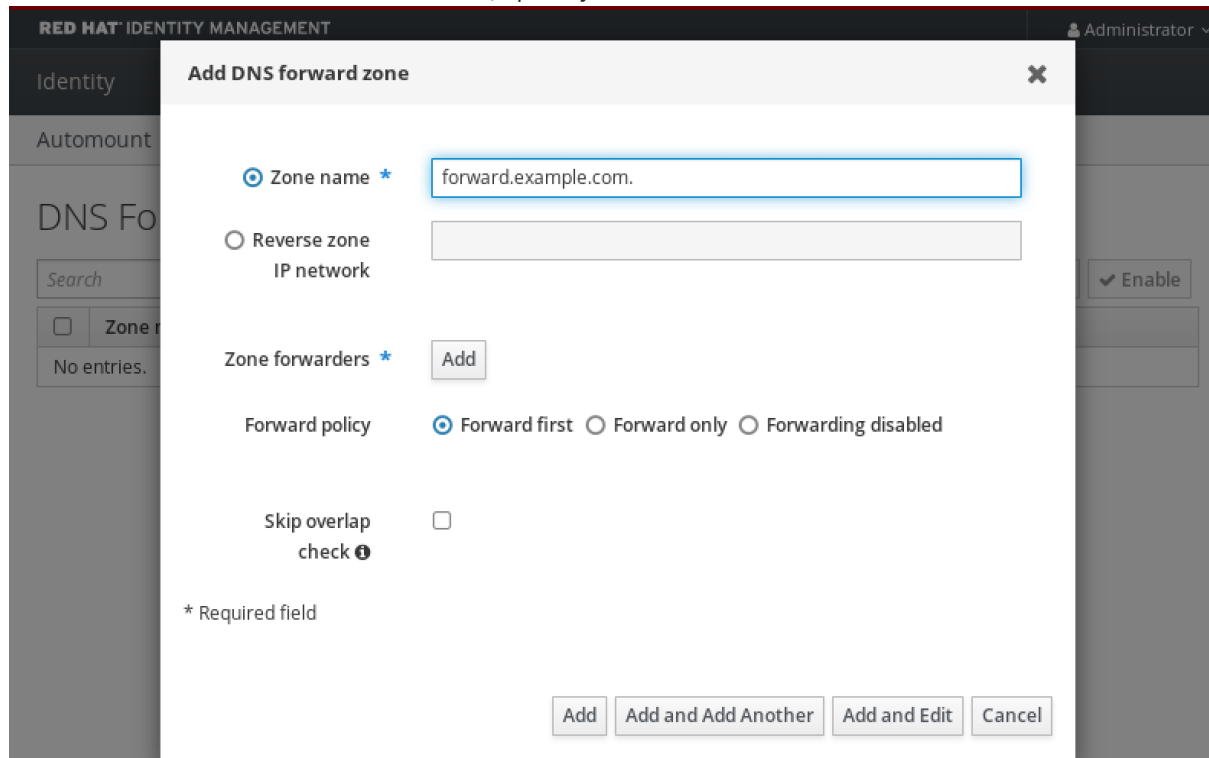
1. In the IdM Web UI, select **Network Services → DNS Forward Zones → DNS**.



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 ☐

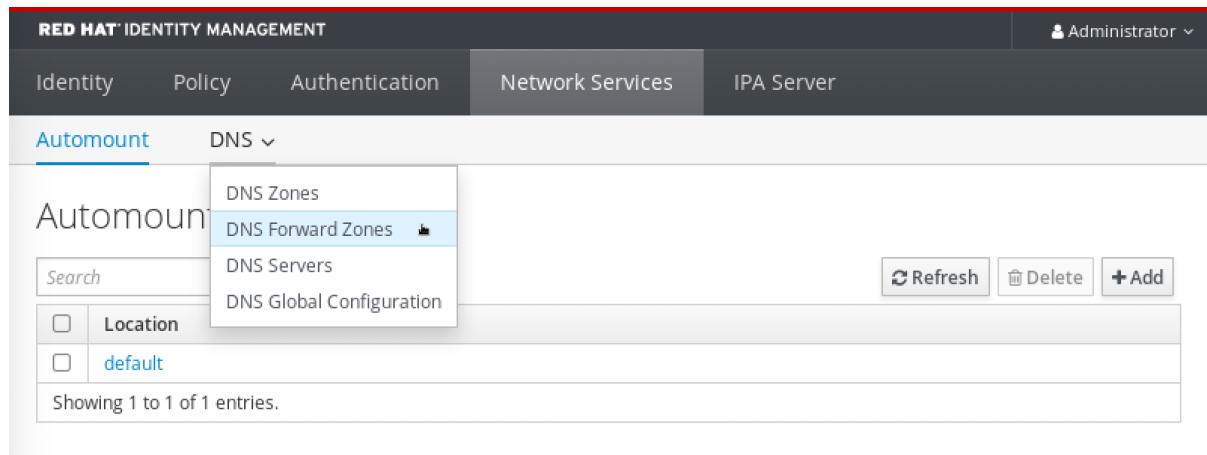
* Required field

Add Add and Add Another Add and Edit Cancel

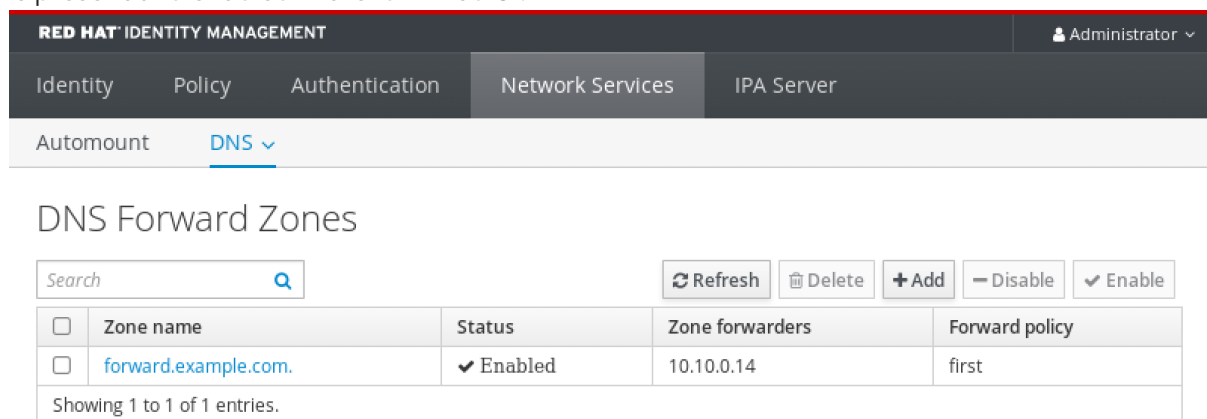
6. Click **Add** at the bottom of the window to add the new forward zone.

Verification

1. In the IdM Web UI, select **Network Services** → **DNS Forward Zones** → **DNS**.

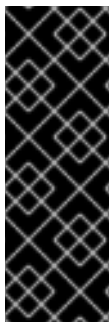


2. Verify that the forward zone you created, with the forwarders and forward policy you specified, is present and enabled in the IdM Web UI.



30.6. ADDING A DNS FORWARD ZONE IN THE CLI

Follow this procedure to add a DNS forward zone by using the command line (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

- 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
```

30.7. ESTABLISHING A DNS GLOBAL FORWARDER IN IDM USING ANSIBLE

Follow this procedure to 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the `secret.yml` Ansible vault stores your `ipadmin_password`.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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 is the modified Ansible playbook file for the current example:

```
---
- name: Playbook to establish a global forwarder in IdM DNS
  hosts: ipaserver

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  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: true
```

6. Save the file.
7. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory.file establish-global-forwarder.yml
```

Additional resources

- The **README-dnsconfig.md** file in the **/usr/share/doc/ansible-freeipa/** directory

30.8. ENSURING THE PRESENCE OF A DNS GLOBAL FORWARDER IN IDM USING ANSIBLE

Follow this procedure to 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the `secret.yml` Ansible vault stores your `ipadmin_password`.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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 is the modified Ansible playbook file for the current example:

```
---
- name: Playbook to ensure the presence of a global forwarder in IdM DNS
  hosts: ipaserver

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  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 --vault-password-file=password_file -v -i inventory.file ensure-presence-
of-a-global-forwarder.yml
```

Additional resources

- The **README-dnsconfig.md** file in the **/usr/share/doc/ansible-freeipa/** directory

30.9. ENSURING THE ABSENCE OF A DNS GLOBAL FORWARDER IN IDM USING ANSIBLE

Follow this procedure to 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 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.

- The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
- The example assumes that the `secret.yml` Ansible vault stores your `ipadmin_password`.
- The target node, that is the node on which the `ansible-freeipa` module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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. Set the `action` variable to **member**.
 - e. Verify the `state` is set to **absent**.

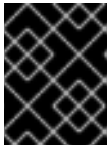
This the modified Ansible playbook file for the current example:

```
---
- name: Playbook to ensure the absence of a global forwarder in IdM DNS
  hosts: ipaserver
```

```

vars_files:
- /home/user_name/MyPlaybooks/secret.yml
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
    action: member
    state: absent

```



IMPORTANT

If you only use the **state: absent** option in your playbook without also using **action: member**, the playbook fails.

6. Save the file.
7. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory.file ensure-absence-of-a-global-forwarder.yml
```

Additional resources

- The **README-dnsconfig.md** file in the `/usr/share/doc/ansible-freeipa/` directory
- The **action: member** option in `ipadnsconfig` [ansible-freeipa](#) modules

30.10. ENSURING DNS GLOBAL FORWARDERS ARE DISABLED IN IDM USING ANSIBLE

Follow this procedure to 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

- 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Disable global forwarders.
    ipadnsconfig:
      forward_policy: none
```

4. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory.file disable-global-forwarders.yml
```

Additional resources

- The **README-dnsconfig.md** file in the **/usr/share/doc/ansible-freeipa/** directory

30.11. ENSURING THE PRESENCE OF A DNS FORWARD ZONE IN IDM USING ANSIBLE

Follow this procedure to 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.

- The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
- The example assumes that the `secret.yml` Ansible vault stores your `ipaadmin_password`.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure the presence of a dnsforwardzone for example.com to 8.8.8.8
    ipadnsforwardzone:
      ipadmin_password: "{{ ipadmin_password }}"
      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 --vault-password-file=password_file -v -i inventory.file ensure-presence-forwardzone.yml
```

Additional resources

- See the **README-dnsforwardzone.md** file in the `/usr/share/doc/ansible-freeipa/` directory.

30.12. ENSURING A DNS FORWARD ZONE HAS MULTIPLE FORWARDERS IN IDM USING ANSIBLE

Follow this procedure to 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.

- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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-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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure presence of 8.8.8.8 and 4.4.4.4 forwarders in dnsforwardzone for
    example.com
    ipadnsforwardzone:
      ipadmin_password: "{{ ipadmin_password }}"
      name: example.com
      forwarders:
        - 8.8.8.8
        - 4.4.4.4
      state: present

```

6. Save the file.

7. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory.file ensure-presence-
multiple-forwarders.yml
```

Additional resources

- See the **README-dnsforwardzone.md** file in the `/usr/share/doc/ansible-freeipa/` directory.

30.13. ENSURING A DNS FORWARD ZONE IS DISABLED IN IDM USING ANSIBLE

Follow this procedure to 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure a dnsforwardzone for example.com is disabled
    ipadnsforwardzone:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: example.com
      state: disabled
```

6. Save the file.
7. Run the playbook:

—

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory.file ensure-disabled-forwardzone.yml
```

Additional resources

- See the **README-dnsforwardzone.md** file in the `/usr/share/doc/ansible-freeipa/` directory.

30.14. ENSURING THE ABSENCE OF A DNS FORWARD ZONE IN IDM USING ANSIBLE

Follow this procedure to 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure the absence of a dnsforwardzone for example.com
    ipadnsforwardzone:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: example.com
      state: absent
```

6. Save the file.
7. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory.file ensure-absence-forwardzone.yml
```

Additional resources

- See the **README-dnsforwardzone.md** file in the **/usr/share/doc/ansible-freeipa/** directory.

CHAPTER 31. 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](#)

31.1. 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.

31.2. COMMON IPA DNSRECORD-* OPTIONS

You can use the following options when adding, modifying and deleting the most common DNS resource record types in 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 31.1. 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 31.2. "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 31.3. "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 31.4. "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.ip6.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 31.5. "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

- Run **ipa dnsrecord-add --help**.

31.3. ENSURING THE PRESENCE OF A AND AAAA DNS RECORDS IN IDM USING ANSIBLE

Follow this procedure to 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the `secret.yml` Ansible vault stores your `ipaadmin_password`.
- The target node, that is the node on which the `ansible-freeipa` module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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 `ipadsrecord` 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:
    ipadmin_password: "{{ ipadmin_password }}"
    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 --vault-password-file=password_file -v -i inventory.file ensure-A-and-AAAA-records-are-present-copy.yml
```

Additional resources

- [DNS records in IdM](#)
- The **README-dnsrecord.md** file in the `/usr/share/doc/ansible-freeipa/` directory
- Sample Ansible playbooks in the `/usr/share/doc/ansible-freeipa/playbooks/dnsrecord` directory

31.4. ENSURING THE PRESENCE OF A AND PTR DNS RECORDS IN IDM USING ANSIBLE

Follow this procedure to 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

- 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 **true**.

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: "{{ ipaadmin_password }}"
        name: host1
        zone_name: idm.example.com
        ip_address: 192.168.122.45
        create_reverse: true
        state: present
```

6. Save the file.
7. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory.file ensure-dnsrecord-with-reverse-is-present-copy.yml
```

Additional resources

- [DNS records in IdM](#)
- The **README-dnsrecord.md** file in the `/usr/share/doc/ansible-freeipa/` directory
- Sample Ansible playbooks in the `/usr/share/doc/ansible-freeipa/playbooks/dnsrecord` directory

31.5. ENSURING THE PRESENCE OF MULTIPLE DNS RECORDS IN IDM USING ANSIBLE

Follow this procedure to 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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:
      ipaadmin_password: "{{ ipaadmin_password }}"
      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 --vault-password-file=password_file -v -i inventory.file ensure-
presence-multiple-records-copy.yml
```

Additional resources

- [DNS records in IdM](#)
- The **README-dnsrecord.md** file in the `/usr/share/doc/ansible-freeipa/` directory
- Sample Ansible playbooks in the `/usr/share/doc/ansible-freeipa/playbooks/dnsrecord` directory

31.6. 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.

Follow this procedure to 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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:
  - ipadnsrecord:
    ipaadmin_password: "{{ ipaadmin_password }}"
    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 --vault-password-file=password_file -v -i inventory.file ensure-
CNAME-record-is-present.yml
```

Additional resources

- See the **README-dnsrecord.md** file in the `/usr/share/doc/ansible-freeipa/` directory.
- See sample Ansible playbooks in the `/usr/share/doc/ansible-freeipa/playbooks/dnsrecord` directory.

31.7. 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.

Follow this procedure to 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 configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the `secret.yml` Ansible vault stores your `ipaadmin_password`.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- 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: "{{ ipaadmin_password }}"
        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 --vault-password-file=password_file -v -i inventory.file ensure-SRV-
record-is-present.yml
```

Additional resources

- [DNS records in IdM](#)
- The **README-dnsrecord.md** file in the **/usr/share/doc/ansible-freeipa/** directory
- Sample Ansible playbooks in the **/usr/share/doc/ansible-freeipa/playbooks/dnsrecord** directory

CHAPTER 32. USING ANSIBLE TO AUTOMOUNT NFS SHARES FOR IDM USERS

Automount is a way to manage, organize, and access directories across multiple systems. Automount automatically mounts a directory whenever access to it is requested. This works well within an Identity Management (IdM) domain as it allows you to share directories on clients within the domain easily.

You can use Ansible to configure NFS shares to be mounted automatically for IdM users logged in to IdM clients in an IdM location.

The example in this chapter uses the following scenario:

- **nfs-server.idm.example.com** is the fully-qualified domain name (FQDN) of a Network File System (NFS) server.
- **nfs-server.idm.example.com** is an IdM client located in the **raleigh** automount location.
- The NFS server exports the **/exports/project** directory as read-write.
- Any IdM user belonging to the **developers** group can access the contents of the exported directory as **/devel/project/** on any IdM client that is located in the same **raleigh** automount location as the NFS server.
- **idm-client.idm.example.com** is an IdM client located in the **raleigh** automount location.



IMPORTANT

If you want to use a Samba server instead of an NFS server to provide the shares for IdM clients, see the Red Hat Knowledgebase solution [How do I configure kerberized CIFS mounts with Autofs in an IPA environment?](#).

32.1. AUTOFS AND AUTOMOUNT IN IDM

The **autofs** service automates the mounting of directories, as needed, by directing the **automount** daemon to mount directories when they are accessed. In addition, after a period of inactivity, **autofs** directs **automount** to unmount auto-mounted directories. Unlike static mounting, on-demand mounting saves system resources.

Automount maps

On a system that utilizes **autofs**, the **automount** configuration is stored in several different files. The primary **automount** configuration file is **/etc/auto.master**, which contains the master mapping of **automount** mount points, and their associated resources, on a system. This mapping is known as *automount maps*.

The **/etc/auto.master** configuration file contains the *master map*. It can contain references to other maps. These maps can either be direct or indirect. Direct maps use absolute path names for their mount points, while indirect maps use relative path names.

Automount configuration in IdM

While **automount** typically retrieves its map data from the local **/etc/auto.master** and associated files, it can also retrieve map data from other sources. One common source is an LDAP server. In the context of Identity Management (IdM), this is a 389 Directory Server.

If a system that uses **autofs** is a client in an IdM domain, the **automount** configuration is not stored in local configuration files. Instead, the **autofs** configuration, such as maps, locations, and keys, is stored

as LDAP entries in the IdM directory. For example, for the **idm.example.com** IdM domain, the default *master map* is stored as follows:

```
dn:
automountmapname=auto.master,cn=default,cn=automount,dc=idm,dc=example,dc=com
objectClass: automountMap
objectClass: top
automountMapName: auto.master
```

Additional resources

- [Mounting file systems on demand](#)

32.2. SETTING UP AN NFS SERVER WITH KERBEROS IN A RED HAT ENTERPRISE LINUX IDENTITY MANAGEMENT DOMAIN

If you use Red Hat Enterprise Linux Identity Management (IdM), you can join your NFS server to the IdM domain. This enables you to centrally manage users and groups and to use Kerberos for authentication, integrity protection, and traffic encryption.

Prerequisites

- The NFS server is [enrolled](#) in a Red Hat Enterprise Linux Identity Management (IdM) domain.
- The NFS server is running and configured.

Procedure

1. Obtain a kerberos ticket as an IdM administrator:

```
# kinit admin
```

2. Create a **nfs/<FQDN>** service principal:

```
# ipa service-add nfs/nfs_server.idm.example.com
```

3. Retrieve the **nfs** service principal from IdM, and store it in the **/etc/krb5.keytab** file:

```
# ipa-getkeytab -s idm_server.idm.example.com -p nfs/nfs_server.idm.example.com -k /etc/krb5.keytab
```

4. Optional: Display the principals in the **/etc/krb5.keytab** file:

```
# klist -k /etc/krb5.keytab
Keytab name: FILE:/etc/krb5.keytab
KVNO Principal
-----
1 nfs/nfs_server.idm.example.com@IDM.EXAMPLE.COM
1 nfs/nfs_server.idm.example.com@IDM.EXAMPLE.COM
1 nfs/nfs_server.idm.example.com@IDM.EXAMPLE.COM
1 nfs/nfs_server.idm.example.com@IDM.EXAMPLE.COM
7 host/nfs_server.idm.example.com@IDM.EXAMPLE.COM
```

```
7 host/nfs_server.idm.example.com@IDM.EXAMPLE.COM
7 host/nfs_server.idm.example.com@IDM.EXAMPLE.COM
7 host/nfs_server.idm.example.com@IDM.EXAMPLE.COM
```

By default, the IdM client adds the host principal to the **/etc/krb5.keytab** file when you join the host to the IdM domain. If the host principal is missing, use the **ipa-getkeytab -s idm_server.idm.example.com -p host/nfs_server.idm.example.com -k /etc/krb5.keytab** command to add it.

5. Use the **ipa-client-automount** utility to configure mapping of IdM IDs:

```
# ipa-client-automount
Searching for IPA server...
IPA server: DNS discovery
Location: default
Continue to configure the system with these values? [no]: yes
Configured /etc/idmapd.conf
Restarting sssd, waiting for it to become available.
Started autofs
```

6. Update your **/etc/exports** file, and add the Kerberos security method to the client options. For example:

```
/nfs/projects/ 192.0.2.0/24(rw,sec=krb5i)
```

If you want that your clients can select from multiple security methods, specify them separated by colons:

```
/nfs/projects/ 192.0.2.0/24(rw,sec=krb5:krb5i:krb5p)
```

7. Reload the exported file systems:

```
# exportfs -r
```

32.3. CONFIGURING AUTOMOUNT LOCATIONS, MAPS, AND KEYS IN IDM BY USING ANSIBLE

As an Identity Management (IdM) system administrator, you can configure automount locations and maps in IdM so that IdM users in the specified locations can access shares exported by an NFS server by navigating to specific mount points on their hosts. Both the exported NFS server directory and the mount points are specified in the maps. In LDAP terms, a location is a container for such map entries.

The example describes how to use Ansible to configure the **raleigh** location and a map that mounts the **nfs-server.idm.example.com:/exports/project** share on the **/devel/project** mount point on the IdM client as a read-write directory.

Prerequisites

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.

- The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
- The example assumes that the `secret.yml` Ansible vault stores your `ipaadmin_password`.
- The target node, that is the node on which the `ansible-freeipa` module is executed, is part of the IdM domain as an IdM client, server or replica.

Procedure

1. On your Ansible control node, navigate to your `~/MyPlaybooks/` directory:

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

2. Copy the `automount-location-present.yml` Ansible playbook file located in the `/usr/share/doc/ansible-freeipa/playbooks/automount/` directory:

```
$ cp /usr/share/doc/ansible-freeipa/playbooks/automount/automount-location-present.yml automount-location-map-and-key-present.yml
```

3. Open the `automount-location-map-and-key-present.yml` file for editing.
4. Adapt the file by setting the following variables in the `ipaautomountlocation` task section:
 - Set the `ipaadmin_password` variable to the password of the IdM `admin`.
 - Set the `name` variable to `raleigh`.
 - Ensure that the `state` variable is set to `present`.
 This is the modified Ansible playbook file for the current example:

```
---
- name: Automount location present example
  hosts: ipaserver
  vars_files:
    - /home/user_name/MyPlaybooks/secret.yml
  tasks:
    - name: Ensure automount location is present
      ipaautomountlocation:
        ipaadmin_password: "{{ ipaadmin_password }}"
        name: raleigh
        state: present
```

5. Continue editing the `automount-location-map-and-key-present.yml` file:
 - a. In the `tasks` section, add a task to ensure the presence of an automount map:

```
[...]
vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
tasks:
[...]
```

```
- name: ensure map named auto.devel in location raleigh is created
  ipaautomountmap:
    ipaadmin_password: "{{ ipaadmin_password }}"
```

```

name: auto.devel
location: raleigh
state: present

```

- b. Add another task to add the mount point and NFS server information to the map:

```

[...]  

vars_files:  

- /home/user_name/MyPlaybooks/secret.yml  

tasks:  

[...]  

- name: ensure automount key /devel/project is present  

  ipaautomountkey:  

    ipaadmin_password: "{{ ipaadmin_password }}"
    location: raleigh
    mapname: auto.devel
    key: /devel/project
    info: nfs-server.idm.example.com:/exports/project
    state: present

```

- c. Add another task to ensure **auto.devel** is connected to **auto.master**:

```

[...]  

vars_files:  

- /home/user_name/MyPlaybooks/secret.yml  

tasks:  

[...]  

- name: Ensure auto.devel is connected in auto.master:  

  ipaautomountkey:  

    ipaadmin_password: "{{ ipaadmin_password }}"
    location: raleigh
    mapname: auto.map
    key: /devel
    info: auto.devel
    state: present

```

6. Save the file.
7. Run the Ansible playbook and specify the playbook and inventory files:

```

$ ansible-playbook --vault-password-file=password_file -v -i inventory automount-
location-map-and-key-present.yml

```

32.4. USING ANSIBLE TO ADD IDM USERS TO A GROUP THAT OWNS NFS SHARES

As an Identity Management (IdM) system administrator, you can use Ansible to create a group of users that is able to access NFS shares, and add IdM users to this group.

This example describes how to use an Ansible playbook to ensure that the **idm_user** account belongs to the **developers** group, so that **idm_user** can access the **/exports/project** NFS share.

Prerequisites

- You have **root** access to the **nfs-server.idm.example.com** NFS server, which is an IdM client located in the **raleigh** automount location.
- On the control node:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the **~/MyPlaybooks/** directory, you have created an **Ansible inventory file** with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
 - In **~/MyPlaybooks/**, you have created the **automount-location-map-and-key-present.yml** file that already contains tasks from [Configuring automount locations, maps, and keys in IdM by using Ansible](#).

Procedure

1. On your Ansible control node, navigate to the **~/MyPlaybooks/** directory:

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

2. Open the **automount-location-map-and-key-present.yml** file for editing.
3. In the **tasks** section, add a task to ensure that the IdM **developers** group exists and **idm_user** is added to this group:

```
[...]
vars_files:
- /home/user_name/MyPlaybooks/secret.yml
tasks:
[...]
- ipagroup:
  ipaadmin_password: "{{ ipaadmin_password }}"
  name: developers
  user:
  - idm_user
  state: present
```

4. Save the file.
5. Run the Ansible playbook and specify the playbook and inventory files:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory automount-
location-map-and-key-present.yml
```

6. On the NFS server, change the group ownership of the **/exports/project** directory to **developers** so that every IdM user in the group can access the directory:

```
# chgrp developers /exports/project
```

32.5. CONFIGURING AUTOMOUNT ON AN IDM CLIENT

As an Identity Management (IdM) system administrator, you can configure automount services on an IdM client so that NFS shares configured for a location to which the client has been added are accessible to an IdM user automatically when the user logs in to the client. The example describes how to configure an IdM client to use automount services that are available in the **raleigh** location.

Prerequisites

- You have **root** access to the IdM client.
- You are logged in as IdM administrator.
- The automount location exists. The example location is **raleigh**.

Procedure

1. On the IdM client, enter the **ipa-client-automount** command and specify the location. Use the **-U** option to run the script unattended:

```
# ipa-client-automount --location raleigh -U
```

2. Stop the autofs service, clear the SSSD cache, and start the autofs service to load the new configuration settings:

```
# systemctl stop autofs ; sss_cache -E ; systemctl start autofs
```

32.6. VERIFYING THAT AN IDM USER CAN ACCESS NFS SHARES ON AN IDM CLIENT

As an Identity Management (IdM) system administrator, you can test if an IdM user that is a member of a specific group can access NFS shares when logged in to a specific IdM client.

In the example, the following scenario is tested:

- An IdM user named **idm_user** belonging to the **developers** group can read and write the contents of the files in the **/devel/project** directory automounted on **idm-client.idm.example.com**, an IdM client located in the **raleigh** automount location.

Prerequisites

- You have [set up an NFS server with Kerberos on an IdM host](#) .
- You have [configured automount locations, maps, and mount points in IdM](#) in which you configured how IdM users can access the NFS share.
- You have [used Ansible to add IdM users to the developers group that owns the NFS shares](#) .
- You have [configured automount on the IdM client](#).

Procedure

1. Verify that the IdM user can access the **read-write** directory:

- a. Connect to the IdM client as the IdM user:

```
$ ssh idm_user@idm-client.idm.example.com
Password:
```

- b. Obtain the ticket-granting ticket (TGT) for the IdM user:

```
$ kinit idm_user
```

- c. Optional: View the group membership of the IdM user:

```
$ ipa user-show idm_user
User login: idm_user
[...]
Member of groups: developers, ipausers
```

- d. Navigate to the **/devel/project** directory:

```
$ cd /devel/project
```

- e. List the directory contents:

```
$ ls
rw_file
```

- f. Add a line to the file in the directory to test the **write** permission:

```
$ echo "idm_user can write into the file" > rw_file
```

- g. Optional: View the updated contents of the file:

```
$ cat rw_file
this is a read-write file
idm_user can write into the file
```

The output confirms that **idm_user** can write into the file.

CHAPTER 33. USING ANSIBLE TO INTEGRATE IDM WITH NIS DOMAINS AND NETGROUPS

33.1. NIS AND ITS BENEFITS

In UNIX environments, the network information service (NIS) is a common way to centrally manage identities and authentication. NIS, which was originally named **Yellow Pages** (YP), centrally manages authentication and identity information such as:

- Users and passwords
- Host names and IP addresses
- POSIX groups

For modern network infrastructures, NIS is considered too insecure because, for example, it neither provides host authentication, nor is data sent encrypted over the network. To work around the problems, NIS is often integrated with other protocols to enhance security.

If you use Identity Management (IdM), you can use the NIS server plug-in to connect clients that cannot be fully migrated to IdM. IdM integrates netgroups and other NIS data into the IdM domain. Additionally, you can easily migrate user and host identities from a NIS domain to IdM.

Netgroups can be used everywhere that NIS groups are expected.

Additional resources

- [NIS in IdM](#)
- [NIS netgroups in IdM](#)
- [Migrating from NIS to Identity Management](#)

33.2. NIS IN IDM

NIS objects in IdM

NIS objects are integrated and stored in the Directory Server back end in compliance with [RFC 2307](#). IdM creates NIS objects in the LDAP directory and clients retrieve them through, for example, System Security Services Daemon (SSSD) or **nss_Idap** using an encrypted LDAP connection.

IdM manages netgroups, accounts, groups, hosts, and other data. IdM uses a NIS listener to map passwords, groups, and netgroups to IdM entries.

NIS Plug-ins in IdM

For NIS support, IdM uses the following plug-ins provided in the **slapi-nis** package:

NIS Server Plug-in

The NIS Server plug-in enables the IdM-integrated LDAP server to act as a NIS server for clients. In this role, Directory Server dynamically generates and updates NIS maps according to the configuration. Using the plug-in, IdM serves clients using the NIS protocol as an NIS server.

Schema Compatibility Plug-in

The Schema Compatibility plug-in enables the Directory Server back end to provide an alternate

view of entries stored in part of the directory information tree (DIT). This includes adding, dropping, or renaming attribute values, and optionally retrieving values for attributes from multiple entries in the tree.

For further details, see the `/usr/share/doc/slapi-nis-version/sch-getting-started.txt` file.

33.3. NIS NETGROUPS IN IDM

NIS entities can be stored in netgroups. Compared to UNIX groups, netgroups provide support for:

- Nested groups (groups as members of other groups).
- Grouping hosts.

A netgroup defines a set of the following information: host, user, and domain. This set is called a **triple**. These three fields can contain:

- A value.
- A dash (-), which specifies "no valid value"
- No value. An empty field specifies a wildcard.

```
(host.example.com,,nisdomain.example.com)
(-,user,nisdomain.example.com)
```

When a client requests a NIS netgroup, IdM translates the LDAP entry :

- To a traditional NIS map and sends it to the client over the NIS protocol by using the NIS plug-in.
- To an LDAP format that is compliant with [RFC 2307](#) or RFC 2307bis.

33.4. USING ANSIBLE TO ENSURE THAT A NETGROUP IS PRESENT

You can use an Ansible playbook to ensure that an IdM netgroup is present. The example describes how to ensure that the **TestNetgroup1** group is present.

Prerequisites

- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package on the Ansible controller.
 - You have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server in the `~/MyPlaybooks/` directory.
 - You have stored your **ipadmin_password** in the **secret.yml** Ansible vault.

Procedure

1. Create your Ansible playbook file **netgroup-present.yml** with the following content:

```
---
```

```

- name: Playbook to manage IPA netgroup.
  hosts: ipaserver
  become: no

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure netgroup members are present
    ipanetgroup:
      ipadmin_password: "{{ ipadmin_password }}"
      name: TestNetgroup1

```

2. Run the playbook:

```

$ ansible-playbook --vault-password-file=password_file -v -i
  path_to_inventory_directory/inventory.file path_to_playbooks_directory/netgroup-
  present.yml

```

Additional resources

- [NIS in IdM](#)
- [/usr/share/doc/ansible-freeipa/README-netgroup.md](#)
- [/usr/share/doc/ansible-freeipa/playbooks/netgroup](#)

33.5. USING ANSIBLE TO ENSURE THAT MEMBERS ARE PRESENT IN A NETGROUP

You can use an Ansible playbook to ensure that IdM users, groups, and netgroups are members of a netgroup. The example describes how to ensure that the **TestNetgroup1** group has the following members:

- The **user1** and **user2** IdM users
- The **group1** IdM group
- The **admins** netgroup
- An **idmclient1** host that is an IdM client

Prerequisites

- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package on the Ansible controller.
 - You have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server in the `~/MyPlaybooks/` directory.
 - You have stored your **ipadmin_password** in the **secret.yml** Ansible vault.
- The **TestNetgroup1** IdM netgroup exists.

- The **user1** and **user2** IdM users exist.
- The **group1** IdM group exists.
- The **admins** IdM netgroup exists.

Procedure

1. Create your Ansible playbook file **IdM-members-present-in-a-netgroup.yml** with the following content:

```
---
- name: Playbook to manage IPA netgroup.
  hosts: ipaserver
  become: no

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure netgroup members are present
    ipanetgroup:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: TestNetgroup1
      user: user1,user2
      group: group1
      host: idmclient1
      netgroup: admins
      action: member
```

2. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i
path_to_inventory_directory/inventory.file path_to_playbooks_directory/IdM-
members-present-in-a-netgroup.yml
```

Additional resources

- [NIS in IdM](#)
- [/usr/share/doc/ansible-freeipa/README-netgroup.md](#)
- [/usr/share/doc/ansible-freeipa/playbooks/netgroup](#)

33.6. USING ANSIBLE TO ENSURE THAT A MEMBER IS ABSENT FROM A NETGROUP

You can use an Ansible playbook to ensure that IdM users are members of a netgroup. The example describes how to ensure that the **TestNetgroup1** group does not have the **user1** IdM user among its members. netgroup

Prerequisites

- You have configured your Ansible control node to meet the following requirements:

- You are using Ansible version 2.14 or later.
- You have installed the [ansible-freeipa](#) package on the Ansible controller.
- You have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server in the `~/MyPlaybooks/` directory.
- You have stored your `ipaadmin_password` in the `secret.yml` Ansible vault.
- The `TestNetgroup1` netgroup exists.

Procedure

1. Create your Ansible playbook file `IdM-member-absent-from-a-netgroup.yml` with the following content:

```
---
- name: Playbook to manage IPA netgroup.
  hosts: ipaserver
  become: no

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure netgroup user, "user1", is absent
    ipanetgroup:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: TestNetgroup1
      user: "user1"
      action: member
      state: absent
```

2. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i
path_to_inventory_directory/inventory.file path_to_playbooks_directory/IdM-
member-absent-from-a-netgroup.yml
```

Additional resources

- [NIS in IdM](#)
- `/usr/share/doc/ansible-freeipa/README-netgroup.md`
- `/usr/share/doc/ansible-freeipa/playbooks/netgroup`

33.7. USING ANSIBLE TO ENSURE THAT A NETGROUP IS ABSENT

You can use an Ansible playbook to ensure that a netgroup does not exist in Identity Management (IdM). The example describes how to ensure that the `TestNetgroup1` group does not exist in your IdM domain.

Prerequisites

- You have configured your Ansible control node to meet the following requirements:

- You are using Ansible version 2.14 or later.
- You have installed the [ansible-freeipa](#) package on the Ansible controller.
- You have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server in the `~/MyPlaybooks/` directory.
- You have stored your `ipaadmin_password` in the `secret.yml` Ansible vault.

Procedure

1. Create your Ansible playbook file `netgroup-absent.yml` with the following content:

```
---
- name: Playbook to manage IPA netgroup.
  hosts: ipaserver
  become: no

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure netgroup my_netgroup1 is absent
    ipanetgroup:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: my_netgroup1
      state: absent
```

2. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i
path_to_inventory_directory/inventory.file path_to_playbooks_directory/netgroup-
absent.yml
```

Additional resources

- [NIS in IdM](#)
- `/usr/share/doc/ansible-freeipa/README-netgroup.md`
- `/usr/share/doc/ansible-freeipa/playbooks/netgroup`

CHAPTER 34. USING ANSIBLE TO CONFIGURE HBAC AND SUDO RULES IN IDM

Using host-based access control (HBAC) in Identity Management (IdM), you can define policies that restrict access to hosts or services based on the following:

- The user attempting to log in and this user's groups
- The host that a user is trying to access and the host groups to which that host belongs
- The service that is being used to access a host

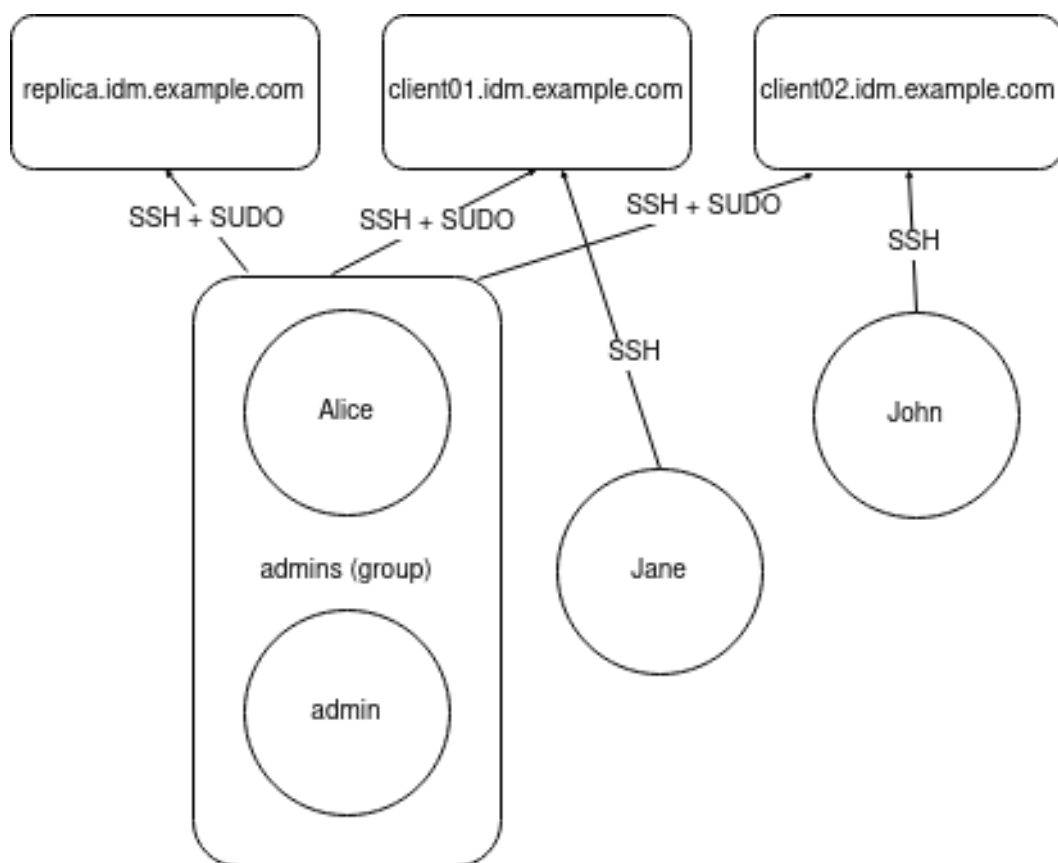
Using **sudo**, a user can run programs as another user, with different privileges, for example **root** privileges. In IdM, you can manage sudo rules centrally. You can define **sudo** rules based on user groups, host groups and command groups, as well as individual users, hosts and commands.

Complete this procedure to ensure the presence of the following HBAC and **sudo** rules for IdM users:

- **jane** can only access host **client01.idm.example.com**.
- **john** can only access host **client02.idm.example.com**.
- Members of the **admins** group, which includes the default **admin** user as well as the regular **alice** user, can access any IdM host.
- Members of the **admins** group can run **sudo** with the following commands on any IdM host:
 - **/usr/sbin/reboot**
 - **/usr/bin/less**
 - **/usr/sbin/setenforce**

The following diagram represents the desired configuration described above:

Figure 34.1. IdM HBAC and SUDO rules diagram

**NOTE**

The procedure illustrates the use of an action group that simplifies the use of modules with **module_defaults**, which is only available in the **freeipa.ansible_freeipa** collection, not the **ansible-freeipa rpm**. You can set default values to be applied to all modules of the collection used in a playbook by using the **action_group** named **freeipa.ansible_freeipa.modules**. In the example, the IdM administrator password is defined in this way. The **ansible-freeipa-collection** is part of the **rhel-9-for-x86_64-appstream-rpms** repository in RHEL 9.5 and later.

Prerequisites

- On the control node:
 - You are using Ansible version 2.14 or later.
 - You are using RHEL version 9.5 or later.
 - You have installed the **ansible-freeipa-collection** subpackage.
 - You have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server in the `~/MyPlaybooks/` directory.
 - You have stored your **ipaadmin_password** in the **secret.yml** Ansible vault.
- The users **jane**, **john** and **alice** exist in IdM. Passwords are configured for these accounts.

Procedure

1. Create your Ansible playbook file **add-hbac-and-sudo-rules-to-idm.yml** with the following content:

■


```

---
- name: Playbook to manage IPA HBAC and SUDO rules
  hosts: ipaserver
  become: false
  gather_facts: false
  collections: freeipa.ansible_freeipa

  vars_files:
  - /home/<user_name>/MyPlaybooks/secret.yml

  module_defaults:
    group/freeipa.ansible_freeipa.modules:
      ipadmin_password: "{{ ipadmin_password }}"

  tasks:
    - name: HBAC Rule for Jane - can log in to client01
      ipahbacrule: # Creates the rule
        name: Jane_rule
        hbacsvc:
          - sshd
          - login
        host: # Host name
          - client01.idm.example.com
        user:
          - jane

    - name: HBAC Rule for John - can log in to client02
      ipahbacrule: # Creates the rule
        name: john_rule
        hbacsvc:
          - sshd
          - login
        host: # Host name
          - client02.idm.example.com
        user:
          - john

    - name: Add user member alice to group admins
      ipagroup:
        name: admins
        action: member
        user:
          - alice

    - name: HBAC Rule for IdM administrators
      ipahbacrule: # Rule to allow admins full access
        name: admin_access # Rule name
        servicecat: all # All services
        hostcat: all # All hosts
        group: # User group
          - admins

    - name: Add reboot command to SUDO
      ipasudocmd:
        name: /usr/sbin/reboot
        state: present

```

```

- name: Add less command to SUDO
  ipasudocmd:
    name: /usr/bin/less
    state: present
- name: Add setenforce command to SUDO
  ipasudocmd:
    name: /usr/sbin/setenforce
    state: present

- name: Create a SUDO command group
  ipasudocmdgroup:
    name: cmd_grp_1
    description: "Group of important commands"
    sudocmd:
      - /usr/sbin/setenforce
      - /usr/bin/less
      - /usr/sbin/reboot
    action: sudocmdgroup
    state: present

- name: Create a SUDO rule with a SUDO command group
  ipasudorule:
    name: sudo_rule_1
    allow_sudocmdgroup:
      - cmd_grp_1
    group: admins
    state: present

- name: Disable allow_all HBAC Rule
  ipahbacrule: # Rule to allow admins full access
    name: allow_all # Rule name
    state: disabled # Disables rule to allow everyone the ability to login

```

2. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -i inventory add-hbac-and-sudo-rules-to-idm.yml
```

Verification

1. Connect to client01 as the jane user:

```

~]$ ssh jane@client01
Password:

Last login: Fri Aug 11 15:32:18 2023 from 192.168.122.1
[jane@client01 ~]$

```

The output verifies that jane has logged in to client01.

2. Try to connect to client02 as the jane user:

```

~]$ ssh jane@client02
Password:
Connection closed by 192.168.122.47 port 22

```

The output verifies that jane cannot log in to client02.

3. Connect to client02 as the alice user:

```
~]$ ssh alice@client02
Password:

Last login: Fri Aug 10 16:13:43 2023 from 192.168.122.1
```

The output verifies that alice has logged in to client02.

4. Try to view the contents of the `/etc/sss/sss.conf` file using **less** without invoking the superuser privileges:

```
[alice@client02 ~]$ less /etc/sss/sss.conf
/etc/sss/sss.conf: Permission denied
```

The attempt fails as the file is not readable by anyone except the owner of the file, which is **root**.

5. Invoke the **root** privileges to view the contents of the `/etc/sss/sss.conf` file using **less**:

```
[alice@client02 ~]$ sudo less /etc/sss/sss.conf
[sudo] password for alice:

[domain/idm.example.com]

id_provider = ipa
ipa_server_mode = True
[...]
```

The output verifies that alice can execute the **less** command on the `/etc/sss/sss.conf` file.

Additional resources

- [Host-based access control rules in IdM](#)
- [Sudo access on an IdM client](#)
- [Using collections in a playbook](#)

CHAPTER 35. USING ANSIBLE TO DELEGATE AUTHENTICATION FOR IDM USERS TO EXTERNAL IDENTITY PROVIDERS

You can use the **idp ansible-freeipa** module to associate users with external identity providers (IdP) that support the OAuth 2 device authorization flow. If an IdP reference and an associated IdP user ID exist, you can use them to enable IdP authentication for an IdM user with the **user ansible-freeipa** module.

Afterward, if these users authenticate with the SSSD version available in RHEL 9.1 or later, they receive RHEL Identity Management (IdM) single sign-on capabilities with Kerberos tickets after performing authentication and authorization at the external IdP.

35.1. THE BENEFITS OF CONNECTING IDM TO AN EXTERNAL IDP

As an administrator, you might want to allow users stored in an external identity source, such as a cloud services provider, to access RHEL systems joined to your Identity Management (IdM) environment. To achieve this, you can delegate the authentication and authorization process of issuing Kerberos tickets for these users to that external entity.

You can use this feature to expand IdM's capabilities and allow users stored in external identity providers (IdPs) to access Linux systems managed by IdM.

35.2. HOW IDM INCORPORATES LOGINS VIA EXTERNAL IDPS

SSSD 2.7.0 contains the **sssd-idp** package, which implements the **idp** Kerberos pre-authentication method. This authentication method follows the OAuth 2.0 Device Authorization Grant flow to delegate authorization decisions to external IdPs:

1. An IdM client user initiates OAuth 2.0 Device Authorization Grant flow, for example, by attempting to retrieve a Kerberos Ticket Granting Ticket (TGT) with the **kinit** utility at the command line.
2. A special code and website link are sent from the Authorization Server to the IdM Key Distribution Center (KDC) backend.
3. The IdM client displays the link and the code to the user. In this example, the IdM client outputs the link and code on the command line.
4. The user opens the website link in a browser, which can be on another host, a mobile phone, and so on:
 - a. The user enters the special code.
 - b. If necessary, the user logs in to the OAuth 2.0-based IdP.
 - c. The user is prompted to authorize the client to access information.
5. The user confirms access at the original device prompt. In this example, the user hits the **Enter** key at the command line.
6. The IdM KDC backend polls the OAuth 2.0 Authorization Server for access to user information.

What is supported:

- Logging in remotely via Secure Shell (SSH) with the **keyboard-interactive** authentication method enabled, which allows calling Pluggable Authentication Module (PAM) libraries.
- Logging in locally with the console via the **logind** service.
- Retrieving a Kerberos TGT with the **kinit** utility.

What is currently not supported:

- Logging in to the IdM WebUI directly. To log in to the IdM WebUI, you must first acquire a Kerberos ticket.
- Logging in to Cockpit WebUI directly. To log in to the Cockpit WebUI, you must first acquire a Kerberos ticket.

Additional resources

- [Authentication against external Identity Providers](#)
- [RFC 8628: OAuth 2.0 Device Authorization Grant](#)

35.3. USING ANSIBLE TO CREATE A REFERENCE TO AN EXTERNAL IDENTITY PROVIDER

To connect external identity providers (IdPs) to your Identity Management (IdM) environment, create IdP references in IdM. Complete this procedure to use the **idp ansible-freeipa** module to configure a reference to the **github** external IdP.

Prerequisites

- You have registered IdM as an OAuth application to your external IdP, and generated a client ID and client secret on the device that an IdM user will be using to authenticate to IdM. The example assumes that:
 - **my_github_account_name** is the github user whose account the IdM user will be using to authenticate to IdM.
 - The **client ID** is **2efe1acffe9e8ab869f4**.
 - The **client secret** is **656a5228abc5f9545c85fa626aecbf69312d398c**.
- Your IdM servers are using RHEL 9.1 or later.
- Your IdM servers are using SSSD 2.7.0 or later.
- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package on the Ansible controller.
 - You are using RHEL 9.4 or later.
 - The example assumes that in the **~/MyPlaybooks/** directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.

- The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.

Procedure

1. On your Ansible control node, create an **configure-external-idp-reference.yml** playbook:

```
---
- name: Configure external IdP
  hosts: ipaserver
  become: false
  gather_facts: false

  tasks:
  - name: Ensure a reference to github external provider is available
    ipaidp:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: github_idp
      provider: github
      client_ID: 2efe1acffe9e8ab869f4
      secret: 656a5228abc5f9545c85fa626aecbf69312d398c
      idp_user_id: my_github_account_name
```

2. Save the file.
3. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory configure-external-idp-reference.yml
```

Verification

- On an IdM client, verify that the output of the **ipa idp-show** command shows the IdP reference you have created.

```
[idmuser@idmclient ~]$ ipa idp-show github_idp
```

Next steps

- [Using Ansible to enable an IdM user to authenticate via an external IdP](#)

Additional resources

- The [idp ansible-freeipa](#) upstream documentation

35.4. USING ANSIBLE TO ENABLE AN IDM USER TO AUTHENTICATE VIA AN EXTERNAL IDP

You can use the **user ansible-freeipa** module to enable an Identity Management (IdM) user to authenticate via an external identity provider (IdP). To do that, associate the external IdP reference you have previously created with the IdM user account. Complete this procedure to use Ansible to associate

an external IdP reference named **github_idp** with the IdM user named **idm-user-with-external-idp**. As a result of the procedure, the user is able to use the **my_github_account_name** github identity to authenticate as **idm-user-with-external-idp** to IdM.

Prerequisites

- Your IdM client and IdM servers are using RHEL 9.1 or later.
- Your IdM client and IdM servers are using SSSD 2.7.0 or later.
- You have created a reference to an external IdP in IdM. See [Using Ansible to create a reference to an external identity provider](#).
- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package on the Ansible controller.
 - You are using RHEL 9.4 or later.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.

Procedure

1. On your Ansible control node, create an **enable-user-to-authenticate-via-external-idp.yml** playbook:

```
---
- name: Ensure an IdM user uses an external IdP to authenticate to IdM
  hosts: ipaserver
  become: false
  gather_facts: false

  tasks:
    - name: Retrieve Github user ID
      ansible.builtin.uri:
        url: "https://api.github.com/users/my_github_account_name"
        method: GET
        headers:
          Accept: "application/vnd.github.v3+json"
      register: user_data

    - name: Ensure IdM user exists with an external IdP authentication
      ipauser:
        ipaadmin_password: "{{ ipaadmin_password }}"
        name: idm-user-with-external-idp
        first: Example
        last: User
        userauthtype: idp
        idp: github_idp
        idp_user_id: my_github_account_name
```

2. Save the file.
3. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory enable-user-to-authenticate-via-external-idp.yml
```

Verification

- Log in to an IdM client and verify that the output of the **ipa user-show** command for the **idm-user-with-external-idp** user displays references to the IdP:

```
$ ipa user-show idm-user-with-external-idp
User login: idm-user-with-external-idp
First name: Example
Last name: User
Home directory: /home/idm-user-with-external-idp
Login shell: /bin/sh
Principal name: idm-user-with-external-idp@idm.example.com
Principal alias: idm-user-with-external-idp@idm.example.com
Email address: idm-user-with-external-idp@idm.example.com
ID: 35000003
GID: 35000003
User authentication types: idp
External IdP configuration: github
External IdP user identifier: idm-user-with-external-idp@idm.example.com
Account disabled: False
Password: False
Member of groups: ipausers
Kerberos keys available: False
```

Additional resources

- The [idp ansible-freeipa](#) upstream documentation

35.5. RETRIEVING AN IDM TICKET-GRANTING TICKET AS AN EXTERNAL IDP USER

If you have delegated authentication for an Identity Management (IdM) user to an external identity provider (IdP), the IdM user can request a Kerberos ticket-granting ticket (TGT) by authenticating to the external IdP.

Complete this procedure to:

1. Retrieve and store an anonymous Kerberos ticket locally.
2. Request the TGT for the **idm-user-with-external-idp** user by using **kinit** with the **-T** option to enable Flexible Authentication via Secure Tunneling (FAST) channel to provide a secure connection between the Kerberos client and Kerberos Distribution Center (KDC).

Prerequisites

- Your IdM client and IdM servers use RHEL 9.1 or later.

- Your IdM client and IdM servers use SSSD 2.7.0 or later.
- You have created a reference to an external IdP in IdM. See [Using Ansible to create a reference to an external identity provider](#).
- You have associated an external IdP reference with the user account. See [Using Ansible to enable an IdM user to authenticate via an external IdP](#).
- The user that you are initially logged in as has write permissions on a directory in the local filesystem.

Procedure

1. Use Anonymous PKINIT to obtain a Kerberos ticket and store it in a file named **./fast.ccache**.

```
$ kinit -n -c ./fast.ccache
```

2. Optional: View the retrieved ticket:

```
$ klist -c fast.ccache
Ticket cache: FILE:fast.ccache
Default principal: WELLKNOWN/ANONYMOUS@WELLKNOWN:ANONYMOUS

Valid starting    Expires          Service principal
03/03/2024 13:36:37 03/04/2024 13:14:28
krbtgt/IDM.EXAMPLE.COM@IDM.EXAMPLE.COM
```

3. Begin authenticating as the IdM user, using the **-T** option to enable the FAST communication channel.

```
[root@client ~]# kinit -T ./fast.ccache idm-user-with-external-idp
Authenticate at https://oauth2.idp.com:8443/auth/realms/master/device?user_code=YHMQ-
XKTL and press ENTER.:
```

4. In a browser, authenticate as the user at the website provided in the command output.
5. At the command line, press the **Enter** key to finish the authentication process.

Verification

- Display your Kerberos ticket information and confirm that the line **config: pa_type** shows **152** for pre-authentication with an external IdP.

```
[root@client ~]# klist -C
Ticket cache: KCM:0:58420
Default principal: idm-user-with-external-idp@IDM.EXAMPLE.COM

Valid starting    Expires          Service principal
05/09/22 07:48:23 05/10/22 07:03:07 krbtgt/IDM.EXAMPLE.COM@IDM.EXAMPLE.COM
config: fast_avail(krbtgt/IDM.EXAMPLE.COM@IDM.EXAMPLE.COM) = yes
08/17/2022 20:22:45 08/18/2022 20:22:43
krbtgt/IDM.EXAMPLE.COM@IDM.EXAMPLE.COM
config: pa_type(krbtgt/IDM.EXAMPLE.COM@IDM.EXAMPLE.COM) = 152
```

The **pa_type = 152** indicates external IdP authentication.

35.6. LOGGING IN TO AN IDM CLIENT VIA SSH AS AN EXTERNAL IDP USER

To log in to an IdM client via SSH as an external identity provider (IdP) user, begin the login process on the command line. When prompted, perform the authentication process at the website associated with the IdP, and finish the process at the Identity Management (IdM) client.

Prerequisites

- Your IdM client and IdM servers are using RHEL 9.1 or later.
- Your IdM client and IdM servers are using SSSD 2.7.0 or later.
- You have created a reference to an external IdP in IdM. See [Using Ansible to create a reference to an external identity provider](#).
- You have associated an external IdP reference with the user account. See [Using Ansible to enable an IdM user to authenticate via an external IdP](#).

Procedure

1. Attempt to log in to the IdM client via SSH.

```
[user@client ~]$ ssh idm-user-with-external-idp@client.idm.example.com
(idm-user-with-external-idp@client.idm.example.com) Authenticate at
https://oauth2.idp.com:8443/auth/realms/main/device?user_code=XYFL-ROYR and press
ENTER.
```

2. In a browser, authenticate as the user at the website provided in the command output.
3. At the command line, press the **Enter** key to finish the authentication process.

Verification

- Display your Kerberos ticket information and confirm that the line **config: pa_type** shows **152** for pre-authentication with an external IdP.

```
[idm-user-with-external-idp@client ~]$ klist -C
Ticket cache: KCM:0:58420
Default principal: idm-user-with-external-idp@IDM.EXAMPLE.COM

Valid starting    Expires          Service principal
05/09/22 07:48:23 05/10/22 07:03:07 krbtgt/IDM.EXAMPLE.COM@IDM.EXAMPLE.COM
config: fast_avail(krbtgt/IDM.EXAMPLE.COM@IDM.EXAMPLE.COM) = yes
08/17/2022 20:22:45 08/18/2022 20:22:43
krbtgt/IDM.EXAMPLE.COM@IDM.EXAMPLE.COM
config: pa_type(krbtgt/IDM.EXAMPLE.COM@IDM.EXAMPLE.COM) = 152
```

35.7. THE PROVIDER OPTION IN THE IPAIDP ANSIBLE MODULE

The following identity providers (IdPs) support OAuth 2.0 device authorization grant flow:

- Microsoft Identity Platform, including Azure AD
- Google
- GitHub
- Keycloak, including Red Hat Single Sign-On (SSO)
- Okta

When using the **idp ansible-freeipa** module to create a reference to one of these external IdPs, you can specify the IdP type with the **provider** option in your **ipaidp ansible-freeipa** playbook task, which expands into additional options as described below:

provider: microsoft

Microsoft Azure IdPs allow parametrization based on the Azure tenant ID, which you can specify with the **organization** option. If you need support for the live.com IdP, specify the option **organization common**.

Choosing **provider: microsoft** expands to use the following options. The value of the **organization** option replaces the string **\${ipaidporg}** in the table.

Option	Value
auth_uri: URI	https://login.microsoftonline.com/\${ipaidporg}/oauth2/v2.0/authorize
dev_auth_uri: URI	https://login.microsoftonline.com/\${ipaidporg}/oauth2/v2.0/devicecode
token_uri: URI	https://login.microsoftonline.com/\${ipaidporg}/oauth2/v2.0/token
userinfo_uri: URI	https://graph.microsoft.com/oidc/userinfo
keys_uri: URI	https://login.microsoftonline.com/common/discovery/v2.0/keys
scope: STR	openid email
idp_user_id: STR	email

provider: google

Choosing **provider: google** expands to use the following options:

Option	Value
auth_uri: URI	https://accounts.google.com/o/oauth2/auth
dev_auth_uri: URI	https://oauth2.googleapis.com/device/code

Option	Value
token_uri: URI	https://oauth2.googleapis.com/token
userinfo_uri: URI	https://openidconnect.googleapis.com/v1/userinfo
keys_uri: URI	https://www.googleapis.com/oauth2/v3/certs
scope: STR	openid email
idp_user_id: STR	email

provider: github

Choosing **provider: github** expands to use the following options:

Option	Value
auth_uri: URI	https://github.com/login/oauth/authorize
dev_auth_uri: URI	https://github.com/login/device/code
token_uri: URI	https://github.com/login/oauth/access_token
userinfo_uri: URI	https://openidconnect.googleapis.com/v1/userinfo
keys_uri: URI	https://api.github.com/user
scope: STR	user
idp_user_id: STR	login

provider: keycloak

With Keycloak, you can define multiple realms or organizations. Since it is often a part of a custom deployment, both base URL and realm ID are required, and you can specify them with the **base_url** and **organization** options in your **ipaidp** playbook task:

```
---
- name: Playbook to manage IPA idp
  hosts: ipaserver
  become: false

  tasks:
  - name: Ensure keycloak idp my-keycloak-idp is present using provider
    ipaidp:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: my-keycloak-idp
      provider: keycloak
```

```

organization: main
base_url: keycloak.domain.com:8443/auth
client_id: my-keycloak-client-id

```

Choosing **provider: keycloak** expands to use the following options. The value you specify in the **base_url** option replaces the string `${ipaidpbaseurl}` in the table, and the value you specify for the **organization** option replaces the string `${ipaidporg}`.

Option	Value
auth_uri: URI	<code>https://\${ipaidpbaseurl}/realms/\${ipaidporg}/protocol/openid-connect/auth</code>
dev_auth_uri: URI	<code>https://\${ipaidpbaseurl}/realms/\${ipaidporg}/protocol/openid-connect/auth/device</code>
token_uri: URI	<code>https://\${ipaidpbaseurl}/realms/\${ipaidporg}/protocol/openid-connect/token</code>
userinfo_uri: URI	<code>https://\${ipaidpbaseurl}/realms/\${ipaidporg}/protocol/openid-connect/userinfo</code>
scope: STR	openid email
idp_user_id: STR	email

provider: okta

After registering a new organization in Okta, a new base URL is associated with it. You can specify this base URL with the **base_url** option in the **ipaidp** playbook task:

```

---
- name: Playbook to manage IPA idp
  hosts: ipaserver
  become: false

  tasks:
  - name: Ensure okta idp my-okta-idp is present using provider
    ipaidp:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: my-okta-idp
      provider: okta
      base_url: dev-12345.okta.com
      client_id: my-okta-client-id

```

Choosing **provider: okta** expands to use the following options. The value you specify for the **base_url** option replaces the string `${ipaidpbaseurl}` in the table.

Option	Value
auth_uri: URI	<code>https://\${ipaidpbaseurl}/oauth2/v1/authorize</code>

Option	Value
dev_auth_uri: URI	https://{ipaidpbaseurl}/oauth2/v1/device/authorize
token_uri: URI	https://{ipaidpbaseurl}/oauth2/v1/token
userinfo_uri: URI	https://{ipaidpbaseurl}/oauth2/v1/userinfo
scope: STR	openid email
idp_user_id: STR	email

Additional resources

- [Pre-populated IdP templates](#)

CHAPTER 36. USING CONSTRAINED DELEGATION IN IDM

Learn more about how you can use the constrained delegation feature in Identity Management (IdM):

- [Constrained delegation in Identity Management](#) describes how constrained delegation works.
- [Configuring a web console to allow a user authenticated with a smart card to SSH to a remote host without being asked to authenticate again](#) describes a use case for constrained delegation in the context of using the Red Hat Enterprise Linux web console to **SSH** to a remote host without requiring authentication.
- [Using Ansible to configure a web console to allow a user authenticated with a smart card to SSH to a remote host without being asked to authenticate again](#) describes a use case for constrained delegation in the context of using Ansible to configure the use of the Red Hat Enterprise Linux web console to **SSH** to a remote host without requiring authentication.
- [Configuring a web console client to allow a user authenticated with a smart card to run sudo without being asked to authenticate](#) describes a use case for constrained delegation in the context of using the Red Hat Enterprise Linux web console to run **sudo** without requiring authentication.
- [Using Ansible to configure a web console to allow a user authenticated with a smart card to run sudo without being asked to authenticate again](#) describes a use case for constrained delegation in the context of using Ansible to configure the use of the Red Hat Enterprise Linux web console to run **sudo** without requiring authentication.

36.1. CONSTRAINED DELEGATION IN IDENTITY MANAGEMENT

The Service for User to Proxy (**S4U2proxy**) extension provides a service that obtains a service ticket to another service on behalf of a user. This feature is known as **constrained delegation**. The second service is typically a proxy performing some work on behalf of the first service, under the authorization context of the user. Using constrained delegation eliminates the need for the user to delegate their full ticket-granting ticket (TGT).

Identity Management (IdM) traditionally uses the Kerberos **S4U2proxy** feature to allow the web server framework to obtain an LDAP service ticket on the user's behalf. The IdM-AD trust system also uses constrained delegation to obtain a **cifs** principal.

You can use the **S4U2proxy** feature to configure a web console client to allow an IdM user that has authenticated with a smart card to achieve the following:

- Run commands with superuser privileges on the RHEL host on which the web console service is running without being asked to authenticate again.
- Access a remote host using **SSH** and access services on the host without being asked to authenticate again.

Additional resources

- [S4U2proxy](#)
- [Service constrained delegation](#)

36.2. CONFIGURING THE WEB CONSOLE TO ALLOW A USER AUTHENTICATED WITH A SMART CARD TO SSH TO A REMOTE HOST WITHOUT BEING ASKED TO AUTHENTICATE AGAIN

After logging in to a user account on the RHEL web console, you can connect to remote machines by using the SSH protocol. You can use the constrained delegation feature to use **SSH** without being asked to authenticate again.

In the example procedure, the web console session runs on the **myhost.idm.example.com** host, and you configure the console to access the **remote.idm.example.com** host by using SSH on behalf of the authenticated user.

Prerequisites

- You have obtained an IdM **admin** ticket-granting ticket (TGT).
- You have **root** access to **remote.idm.example.com**.
- The **cockpit** service is running in IdM.
- The web console has created an **S4U2Proxy** Kerberos ticket in the user session. To verify it, log in to the web console as an IdM user, open the **Terminal** page, and enter:

```
$ klist
Ticket cache: FILE:/run/user/1894000001/cockpit-session-3692.ccache
Default principal: user@IDM.EXAMPLE.COM

Valid starting    Expires          Service principal
07/30/21 09:19:06 07/31/21 09:19:06
HTTP/myhost.idm.example.com@IDM.EXAMPLE.COM
07/30/21 09:19:06 07/31/21 09:19:06 krbtgt/IDM.EXAMPLE.COM@IDM.EXAMPLE.COM
                    for client HTTP/myhost.idm.example.com@IDM.EXAMPLE.COM
```

Procedure

1. Create a list of the target hosts that the delegation rule can access:

- a. Create a service delegation target:

```
$ ipa servicedelegationtarget-add cockpit-target
```

- b. Add the target host to the delegation target:

```
$ ipa servicedelegationtarget-add-member cockpit-target \
--principals=host/remote.idm.example.com@IDM.EXAMPLE.COM
```

2. Allow **cockpit** sessions to access the target host list by creating a service delegation rule and adding the HTTP service Kerberos principal to it:

- a. Create a service delegation rule:

```
$ ipa servicedelegationrule-add cockpit-delegation
```

- b. Add the web console client to the delegation rule:


```
$ ipa servicedelegationrule-add-member cockpit-delegation \
  --principals=HTTP/myhost.idm.example.com@IDM.EXAMPLE.COM
```

- c. Add the delegation target to the delegation rule:

```
$ ipa servicedelegationrule-add-target cockpit-delegation \
  --servicedelegationtargets=cockpit-target
```

3. Enable Kerberos authentication on the **remote.idm.example.com** host:
 - a. Connect through SSH to **remote.idm.example.com** as **root**.
 - b. Open the **/etc/ssh/sshd_config** file for editing.
 - c. Enable **GSSAPIAuthentication** by uncommenting the **GSSAPIAuthentication no** line and replacing it with **GSSAPIAuthentication yes**.
4. Restart the **sshd** service on **remote.idm.example.com** so that the changes take effect immediately:

```
$ systemctl try-restart sshd.service
```

Additional resources

- [Logging in to the web console with smart cards](#)
- [Constrained delegation in Identity Management](#)

36.3. USING ANSIBLE TO CONFIGURE THE WEB CONSOLE TO ALLOW A USER AUTHENTICATED WITH A SMART CARD TO SSH TO A REMOTE HOST WITHOUT BEING ASKED TO AUTHENTICATE AGAIN

After logging in to a user account on the RHEL web console, you can connect to remote machines by using the SSH protocol. You can use the **servicedelegationrule** and **servicedelegationtarget** modules to configure the web console for the constrained delegation feature, which enables SSH connections without being asked to authenticate again.

In the example procedure, the web console session runs on the **myhost.idm.example.com** host and you configure it to access the **remote.idm.example.com** host by using SSH on behalf of the authenticated user.

Prerequisites

- The IdM **admin** password.
- **root** access to **remote.idm.example.com**.
- The web console service runs in IdM.
- The web console has created an **S4U2Proxy** Kerberos ticket in the user session. To verify it, log in to the web console as an IdM user, open the **Terminal** page, and enter:

```
$ klist
Ticket cache: FILE:/run/user/1894000001/cockpit-session-3692.ccache
```

```
Default principal: user@IDM.EXAMPLE.COM
```

```
Valid starting    Expires          Service principal
07/30/21 09:19:06 07/31/21 09:19:06
HTTP/myhost.idm.example.com@IDM.EXAMPLE.COM
07/30/21 09:19:06 07/31/21 09:19:06 krbtgt/IDM.EXAMPLE.COM@IDM.EXAMPLE.COM
for client HTTP/myhost.idm.example.com@IDM.EXAMPLE.COM
```

- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the **ansible-freeipa** package.
 - The example assumes that in the `~/MyPlaybooks/` directory, you have created an [Ansible inventory file](#) with the fully-qualified domain name (FQDN) of the IdM server.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

Procedure

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

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

2. Create a **web-console-smart-card-ssh.yml** playbook with the following content:
 - a. Create a task that ensures the presence of a delegation target:

```
---
- name: Playbook to create a constrained delegation target
  hosts: ipaserver

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml

  tasks:
  - name: Ensure servicedelegationtarget web-console-delegation-target is present
    ipaservicedelegationtarget:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: web-console-delegation-target
```

- b. Add a task that adds the target host to the delegation target:

```
- name: Ensure servicedelegationtarget web-console-delegation-target member
principal host/remote.idm.example.com@IDM.EXAMPLE.COM is present
ipaservicedelegationtarget:
  ipaadmin_password: "{{ ipaadmin_password }}"
  name: web-console-delegation-target
  principal: host/remote.idm.example.com@IDM.EXAMPLE.COM
  action: member
```

- c. Add a task that ensures the presence of a delegation rule:

```
- name: Ensure servicedelegationrule delegation-rule is present
  ipaservicedelegationrule:
    ipaadmin_password: "{{ ipaadmin_password }}"
    name: web-console-delegation-rule
```

- d. Add a task that ensures that the Kerberos principal of the web console client service is a member of the constrained delegation rule:

```
- name: Ensure the Kerberos principal of the web console client service is added to the
  servicedelegationrule web-console-delegation-rule
  ipaservicedelegationrule:
    ipaadmin_password: "{{ ipaadmin_password }}"
    name: web-console-delegation-rule
    principal: HTTP/myhost.idm.example.com
    action: member
```

- e. Add a task that ensures that the constrained delegation rule is associated with the web-console-delegation-target delegation target:

```
- name: Ensure a constrained delegation rule is associated with a specific delegation
  target
  ipaservicedelegationrule:
    ipaadmin_password: "{{ ipaadmin_password }}"
    name: web-console-delegation-rule
    target: web-console-delegation-target
    action: member
```

3. Save the file.
4. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory web-console-smart-
card-ssh.yml
```

5. Enable Kerberos authentication on **remote.idm.example.com**:
 - a. Connect through SSH to **remote.idm.example.com** as **root**.
 - b. Open the **/etc/ssh/sshd_config** file for editing.
 - c. Enable **GSSAPIAuthentication** by uncommenting the **GSSAPIAuthentication no** line and replacing it with **GSSAPIAuthentication yes**.
6. Restart the **sshd** service on **remote.idm.example.com** so that the changes take effect immediately:

```
$ systemctl try-restart sshd.service
```

Additional resources

- [Logging in to the web console with smart cards](#)
- [Constrained delegation in Identity Management](#)

- **README-servicedelegationrule.md** and **README-servicedelegationtarget.md** in the `/usr/share/doc/ansible-freeipa/` directory
- Sample playbooks in the `/usr/share/doc/ansible-freeipa/playbooks/servicedelegationtarget` and `/usr/share/doc/ansible-freeipa/playbooks/servicedelegationrule` directories

36.4. CONFIGURING A WEB CONSOLE TO ALLOW A USER AUTHENTICATED WITH A SMART CARD TO RUN SUDO WITHOUT BEING ASKED TO AUTHENTICATE AGAIN

After you have logged in to a user account on the RHEL web console, as an Identity Management (IdM) system administrator you might need to run commands with superuser privileges. You can use the [constrained delegation](#) feature to run **sudo** on the system without being asked to authenticate again.

Follow this procedure to configure a web console to use constrained delegation. In the example below, the web console session runs on the **myhost.idm.example.com** host.

Prerequisites

- You have obtained an IdM **admin** ticket-granting ticket (TGT).
- The web console service is present in IdM.
- The **myhost.idm.example.com** host is present in IdM.
- You enabled **admin sudo** access to domain administrators on the IdM server. For details, see [Enabling sudo access for IdM administrators on IdM hosts](#).
- The web console has created an **S4U2Proxy** Kerberos ticket in the user session. To verify that this is the case, log in to the web console as an IdM user, open the **Terminal** page, and enter:

```
$ klist
Ticket cache: FILE:/run/user/1894000001/cockpit-session-3692.ccache
Default principal: user@IDM.EXAMPLE.COM

Valid starting    Expires          Service principal
07/30/21 09:19:06 07/31/21 09:19:06
HTTP/myhost.idm.example.com@IDM.EXAMPLE.COM
07/30/21 09:19:06 07/31/21 09:19:06 krbtgt/IDM.EXAMPLE.COM@IDM.EXAMPLE.COM
                  for client HTTP/myhost.idm.example.com@IDM.EXAMPLE.COM
```

Procedure

1. Create a list of the target hosts that can be accessed by the delegation rule:
 - a. Create a service delegation target:

```
$ ipa servicedelegationtarget-add cockpit-target
```

- b. Add the target host to the delegation target:

```
$ ipa servicedelegationtarget-add-member cockpit-target \
--principals=host/myhost.idm.example.com@IDM.EXAMPLE.COM
```

2. Allow **cockpit** sessions to access the target host list by creating a service delegation rule and adding the **HTTP** service Kerberos principal to it:

- a. Create a service delegation rule:

```
$ ipa servicedelegationrule-add cockpit-delegation
```

- b. Add the web console service to the delegation rule:

```
$ ipa servicedelegationrule-add-member cockpit-delegation \
--principals=HTTP/myhost.idm.example.com@IDM.EXAMPLE.COM
```

- c. Add the delegation target to the delegation rule:

```
$ ipa servicedelegationrule-add-target cockpit-delegation \
--servicedelegationtargets=cockpit-target
```

3. Enable **pam_sss_gss**, the PAM module for authenticating users over the Generic Security Service Application Program Interface (GSSAPI) in cooperation with the System Security Services Daemon (SSSD):

- a. Open the **/etc/sss/sss.conf** file for editing.

- b. Specify that **pam_sss_gss** can provide authentication for the **sudo** and **sudo -i** commands in IdM your domain:

```
[domain/idm.example.com]
pam_gssapi_services = sudo, sudo-i
```

- c. Save and exit the file.

- d. Open the **/etc/pam.d/sudo** file for editing.

- e. Insert the following line to the top of the **#%PAM-1.0** list to allow, but not require, GSSAPI authentication for **sudo** commands:

```
auth sufficient pam_sss_gss.so
```

- f. Save and exit the file.

4. Restart the **SSSD** service so that the above changes take effect immediately:

```
$ systemctl restart sssd
```

Additional resources

- [Logging in to the web console with smart cards](#)
- [Constrained delegation in Identity Management](#)

36.5. USING ANSIBLE TO CONFIGURE A WEB CONSOLE TO ALLOW A USER AUTHENTICATED WITH A SMART CARD TO RUN SUDO WITHOUT BEING ASKED TO AUTHENTICATE AGAIN

After you have logged in to a user account on the RHEL web console, as an Identity Management (IdM) system administrator you might need to run commands with superuser privileges. You can use the [constrained delegation](#) feature to run **sudo** on the system without being asked to authenticate again.

Follow this procedure to use the **ipaservicedelegationrule** and **ipaservicedelegationtarget ansible-freeipa** modules to configure a web console to use constrained delegation. In the example below, the web console session runs on the **myhost.idm.example.com** host.

Prerequisites

- You have obtained an IdM **admin** ticket-granting ticket (TGT) by authenticating to the web console session with a smart card..
- The web console service has been enrolled into IdM.
- The **myhost.idm.example.com** host is present in IdM.
- You enabled **admin sudo** access to domain administrators on the IdM server. For details, see [Enabling sudo access for IdM administrators on IdM hosts](#) .
- The web console has created an **S4U2Proxy** Kerberos ticket in the user session. To verify that this is the case, log in to the web console as an IdM user, open the **Terminal** page, and enter:

```
$ klist
```

```
Ticket cache: FILE:/run/user/1894000001/cockpit-session-3692.ccache
```

```
Default principal: user@IDM.EXAMPLE.COM
```

```
Valid starting    Expires          Service principal
```

```
07/30/21 09:19:06 07/31/21 09:19:06
```

```
HTTP/myhost.idm.example.com@IDM.EXAMPLE.COM
```

```
07/30/21 09:19:06 07/31/21 09:19:06 krbtgt/IDM.EXAMPLE.COM@IDM.EXAMPLE.COM
```

```
for client HTTP/myhost.idm.example.com@IDM.EXAMPLE.COM
```

- You have configured your Ansible control node to meet the following requirements:
 - You are using Ansible version 2.14 or later.
 - You have installed the [ansible-freeipa](#) package.
 - The example assumes that 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 the constrained delegation.
 - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.

Procedure

1. On your Ansible control node, navigate to your **~/MyPlaybooks/** directory:

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

2. Create a **web-console-smart-card-sudo.yml** playbook with the following content:

```
- Create a task that ensures the presence of a delegation target:
```

- a. Create a task that ensures the presence of a delegation target:

```
---
- name: Playbook to create a constrained delegation target
  hosts: ipaserver

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Ensure servicedelegationtarget named sudo-web-console-delegation-target is
    present
    ipaservicedelegationtarget:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: sudo-web-console-delegation-target
```

- b. Add a task that adds the target host to the delegation target:

```
- name: Ensure that a member principal named
host/myhost.idm.example.com@IDM.EXAMPLE.COM is present in a service delegation
target named sudo-web-console-delegation-target
ipaservicedelegationtarget:
  ipaadmin_password: "{{ ipaadmin_password }}"
  name: sudo-web-console-delegation-target
  principal: host/myhost.idm.example.com@IDM.EXAMPLE.COM
  action: member
```

- c. Add a task that ensures the presence of a delegation rule:

```
- name: Ensure servicedelegationrule named sudo-web-console-delegation-rule is
present
ipaservicedelegationrule:
  ipaadmin_password: "{{ ipaadmin_password }}"
  name: sudo-web-console-delegation-rule
```

- d. Add a task that ensures that the Kerberos principal of the web console service is a member of the constrained delegation rule:

```
- name: Ensure the Kerberos principal of the web console service is added to the
service delegation rule named sudo-web-console-delegation-rule
ipaservicedelegationrule:
  ipaadmin_password: "{{ ipaadmin_password }}"
  name: sudo-web-console-delegation-rule
  principal: HTTP/myhost.idm.example.com
  action: member
```

- e. Add a task that ensures that the constrained delegation rule is associated with the sudo-web-console-delegation-target delegation target:

```
- name: Ensure a constrained delegation rule is associated with a specific delegation
target
ipaservicedelegationrule:
  ipaadmin_password: "{{ ipaadmin_password }}"
  name: sudo-web-console-delegation-rule
  target: sudo-web-console-delegation-target
  action: member
```

3. Save the file.
4. Run the Ansible playbook. Specify the playbook file, the file storing the password protecting the **secret.yml** file, and the inventory file:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory web-console-smart-card-sudo.yml
```

5. Enable **pam_sss_gss**, the PAM module for authenticating users over the Generic Security Service Application Program Interface (GSSAPI) in cooperation with the System Security Services Daemon (SSSD):

- a. Open the **/etc/sss/sss.conf** file for editing.
- b. Specify that **pam_sss_gss** can provide authentication for the **sudo** and **sudo -i** commands in IdM your domain:

```
[domain/idm.example.com]  
pam_gssapi_services = sudo, sudo-i
```

- c. Save and exit the file.
- d. Open the **/etc/pam.d/sudo** file for editing.
- e. Insert the following line to the top of the **#%PAM-1.0** list to allow, but not require, GSSAPI authentication for **sudo** commands:

```
auth sufficient pam_sss_gss.so
```

- f. Save and exit the file.
6. Restart the **SSSD** service so that the above changes take effect immediately:

```
$ systemctl restart sssd
```

Additional resources

- [Constrained delegation in Identity Management](#)
- **README-servicedelegationrule.md** and **README-servicedelegationtarget.md** in the **/usr/share/doc/ansible-freeipa/** directory
- Sample playbooks in the **/usr/share/doc/ansible-freeipa/playbooks/servicedelegationtarget** and **/usr/share/doc/ansible-freeipa/playbooks/servicedelegationrule** directories

36.6. ADDITIONAL RESOURCES

- [Managing remote systems in the web console](#)

CHAPTER 37. JOINING RHEL SYSTEMS TO AN ACTIVE DIRECTORY BY USING RHEL SYSTEM ROLES

If your organization uses Microsoft Active Directory (AD) to centrally manage users, groups, and other resources, you can join your Red Hat Enterprise Linux (RHEL) host to this AD. For example, AD users can then log into RHEL and you can make services on the RHEL host available for authenticated AD users. By using the **ad_integration** RHEL system role, you can automate the integration of Red Hat Enterprise Linux system into an Active Directory (AD) domain.



NOTE

The **ad_integration** role is for deployments using direct AD integration without an Identity Management (IdM) environment. For IdM environments, use the **ansible-freeipa** roles.

The **ad_integration** system role is not included in the **ansible-freeipa** package. It is part of the **rhel-system-roles** package. You can install **rhel-system-roles** on systems with a **Red Hat Enterprise Linux Server** subscription attached.

37.1. JOINING RHEL TO AN ACTIVE DIRECTORY DOMAIN BY USING THE **AD_INTEGRATION** RHEL SYSTEM ROLE

You can use the **ad_integration** RHEL system role to automate the process of joining RHEL to an Active Directory (AD) domain.

Prerequisites

- [You have prepared the control node and the managed nodes](#)
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.
- The managed node uses a DNS server that can resolve AD DNS entries.
- Credentials of an AD account which has permissions to join computers to the domain.
- Ensure that the required ports are open:
 - [Ports required for direct integration of RHEL systems into AD using SSSD](#)

Procedure

1. Store your sensitive variables in an encrypted file:
 - a. Create the vault:

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

- b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the **<key>: <value>** format:

```
usr: administrator
pwd: <password>
```

- c. Save the changes, and close the editor. Ansible encrypts the data in the vault.
2. Create a playbook file, for example `~/playbook.yml`, with the following content:

```
---
- name: Active Directory integration
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Join an Active Directory
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.ad_integration
      vars:
        ad_integration_user: "{{ usr }}"
        ad_integration_password: "{{ pwd }}"
        ad_integration_realm: "ad.example.com"
        ad_integration_allow_rc4_crypto: false
        ad_integration_timesync_source: "time_server.ad.example.com"
```

The settings specified in the example playbook include the following:

ad_integration_allow_rc4_crypto: <true/false>

Configures whether the role activates the **AD-SUPPORT** crypto policy on the managed node. By default, RHEL does not support the weak RC4 encryption but, if Kerberos in your AD still requires RC4, you can enable this encryption type by setting

ad_integration_allow_rc4_crypto: true.

Omit this the variable or set it to **false** if Kerberos uses AES encryption.

ad_integration_timesync_source: <time_server>

Specifies the NTP server to use for time synchronization. Kerberos requires a synchronized time among AD domain controllers and domain members to prevent replay attacks. If you omit this variable, the **ad_integration** role does not utilize the **timesync** RHEL system role to configure time synchronization on the managed node.

For details about all variables used in the playbook, see the `/usr/share/ansible/roles/rhel-system-roles.ad_integration/README.md` file on the control node.

3. Validate the playbook syntax:

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

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

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

Verification

- Check if AD users, such as **administrator**, are available locally on the managed node:

```
$ ansible managed-node-01.example.com -m command -a 'getent passwd  
administrator@ad.example.com'  
administrator@ad.example.com:*:1450400500:1450400513:Administrator:/home/administrator  
@ad.example.com:/bin/bash
```

Additional resources

- `/usr/share/ansible/roles/rhel-system-roles.ad_integration/README.md` file
- `/usr/share/doc/rhel-system-roles/ad_integration/` directory
- [Ansible vault](#)