



Practice
Tests



Flash
Cards



Labs



Study
Planner

Cert Guide

Advance your IT career with hands-on learning

Red Hat RHCE™ 8 (EX294)



SANDER VAN VUGT

FREE SAMPLE CHAPTER

SHARE WITH OTHERS



Red Hat RHCE™ 8 (EX294) Cert Guide

Sander van Vugt



Pearson
221 River Street
Hoboken, NJ 07030 USA

Red Hat RHCE 8 (EX294) Cert Guide

Copyright © 2021 by Pearson Education, Inc.

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-0-13-687243-6

ISBN-10: 0-13-687243-3

Library of Congress Control Number: 2020941743

ScoutAutomatedPrintCode

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Pearson IT Certification cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Pearson IT Certification and Sander van Vugt have no affiliation with Red Hat, Inc. The RED HAT and RHCE trademarks are used for identification purposes only and are not intended to indicate affiliation with or approval by Red Hat, Inc.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Special Sales

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

Editor-in-Chief

Mark Taub

Product Line Manager

Brett Bartow

Executive Editor

Denise Lincoln

Development Editor

Ellie Bru

Managing Editor

Sandra Schroeder

Senior Project Editor

Tonya Simpson

Copy Editor

Chuck Hutchinson

Indexer

Timothy Wright

Proofreader

Donna Mulder

Technical Editors

John McDonough

William “Bo” Rothwell

Publishing Coordinator

Cindy Teeters

Cover Designer

Chuti Prasertsith

Compositor

codeMantra

Contents at a Glance

Introduction xxii

Part I: Introduction to Ansible

- CHAPTER 1 Understanding Configuration Management 3
- CHAPTER 2 Installing Ansible 15
- CHAPTER 3 Setting Up an Ansible Managed Environment 31
- CHAPTER 4 Using Ad Hoc Commands 47
- CHAPTER 5 Getting Started with Playbooks 69
- CHAPTER 6 Working with Variables and Facts 97

Part II: Common Ansible Management Tasks

- CHAPTER 7 Using Task Control 131
- CHAPTER 8 Deploying Files 173
- CHAPTER 9 Using Ansible Roles 205
- CHAPTER 10 Using Ansible in Large Environments 229
- CHAPTER 11 Troubleshooting Ansible 251

Part III: Managing Systems with Ansible

- CHAPTER 12 Managing Software with Ansible 281
- CHAPTER 13 Managing Users 305
- CHAPTER 14 Managing Services and the Boot Process 333
- CHAPTER 15 Managing Storage 351
- CHAPTER 16 Final Preparation 383

Part IV: Practice Exams

- Practice Exam A 387
- Practice Exam B 391

Part V: Appendixes

- APPENDIX A Answers to the “Do I Know This Already?” Quizzes and Review Questions 395
- APPENDIX B Getting Started with Ansible Tower 415

APPENDIX C *Red Hat RHCE 8 (EX294) Cert Guide Exam Updates* 431

Glossary 433

Index 439

Online Elements:

Practice Exam C

Practice Exam D

APPENDIX D Memory Tables

APPENDIX E Memory Tables Answer Key

APPENDIX F Study Planner

Glossary

Table of Contents

Introduction xxii

Part I: Introduction to Ansible

Chapter 1 Understanding Configuration Management 3

“Do I Know This Already?” Quiz 3

Foundation Topics 6

Understanding Automation 6

What Is Automation? 6

Understanding the DevOps Way of Working 7

Understanding Infrastructure as Code 8

Other Automation Solutions 8

Understanding Ansible Essential Components 9

Ansible Is Python 9

Ansible Architecture 9

Understanding Ansible Tower 10

Understanding the Ansible Way 11

Understanding Ansible Use Cases 11

Using Ansible for Configuration Management 11

Using Ansible for Provisioning 12

Using Ansible for Continuous Delivery 12

Summary 12

Exam Preparation Tasks 12

Review All Key Topics 12

Define Key Terms 13

Chapter 2 Installing Ansible 15

“Do I Know This Already?” Quiz 15

Foundation Topics 18

Understanding an Ansible Environment 18

Understanding Controller Host Requirements 20

Understanding Installation Methods 20

Installing Ansible on RHEL 8 21

Installing Ansible on CentOS 8 21

Using python-pip to Install Ansible 21

Configuring Managed Hosts	22
Configuring the Ansible User	23
Setting Up SSH for the Ansible User	23
Configuring sudo for the Ansible User	25
Summary	26
Exam Preparation Tasks	26
Review All Key Topics	26
Memory Tables	27
Define Key Terms	27
Review Questions	27
End-of-Chapter Lab	28
Lab 2-1	28
Chapter 3	Setting Up an Ansible Managed Environment 31
“Do I Know This Already?” Quiz	31
Foundation Topics	34
Understanding Projects	34
Configuring Static Inventory	35
Listing Hosts	35
Inventory Host Groups	36
Using Inventory in Commands	37
Specifying Host Variables	38
Working with Dynamic Inventory	39
Working with Community-Provided Dynamic Inventory Scripts	39
Writing Your Own Inventory Scripts	40
Using Multiple Inventory Files	42
Managing Settings in <code>ansible.cfg</code>	42
Summary	44
Exam Preparation Tasks	44
Review All Key Topics	44
Memory Tables	44
Define Key Terms	44
Review Questions	45
End-of-Chapter Lab	45
Lab 3-1	45

Chapter 4	Using Ad Hoc Commands	47
	“Do I Know This Already?” Quiz	47
	Foundation Topics	50
	Understanding Ad Hoc Commands	50
	Working with Modules	53
	Exploring Essential Modules	54
	<i>command</i>	54
	<i>shell</i>	55
	<i>raw</i>	55
	<i>copy</i>	55
	<i>yum</i>	56
	<i>service</i>	56
	<i>ping</i>	56
	Browsing Available Modules	58
	Consulting Module Documentation	58
	Using <code>ansible-doc</code>	58
	Using <code>https://docs.ansible.com</code>	62
	Running Ad Hoc Commands from Shell Scripts	64
	Summary	66
	Exam Preparation Tasks	66
	Review All Key Topics	66
	Memory Tables	66
	Define Key Terms	66
	Review Questions	67
	End-of-Chapter Lab	67
	Lab 4-1	67
Chapter 5	Getting Started with Playbooks	69
	“Do I Know This Already?” Quiz	69
	Foundation Topics	72
	Exploring Your First Playbook	72
	From Ad Hoc Commands to Playbook	72
	Playbook Elements	73
	Running the Playbook	74
	Undoing Playbook Modifications	75

Working with YAML	77
Indentation	77
Using Key-Value Pairs	78
Understanding YAML Lists	79
Using YAML Strings	79
Verifying Syntax	80
Performing a Playbook Dry Run	81
Managing Multiplay Playbooks	83
Multiplay Playbook Considerations	83
Multiplay Playbook Example	84
Increasing Output Verbosity	86
Summary	92
Exam Preparation Tasks	92
Review All Key Topics	93
Memory Tables	93
Define Key Terms	93
Review Questions	93
End-of-Chapter Lab	94
Lab 5-1	94
Chapter 6 Working with Variables and Facts	97
“Do I Know This Already?” Quiz	97
Foundation Topics	100
Understanding the Use of Variables in Ansible Playbooks	100
Working with Ansible Facts	101
Gathering Facts	102
Understanding How Facts Are Displayed	104
Managing Fact Gathering	108
Working with Custom Facts	109
Working with Variables	111
Defining Variables	112
Using Include Files	112
Managing Host and Group Variables	113
Using Multivalued Variables	115
Working with Magic Variables	117
Understanding Variable Precedence	119

Using Vault to Manage Sensitive Values	119
Understanding Vault	119
Managing Encrypted Files	120
Using Vault in Playbooks	120
Managing Files with Sensitive Variables	121
Capturing Command Output Using register	122
Summary	125
Exam Preparation Tasks	125
Review All Key Topics	125
Memory Tables	125
Define Key Terms	126
Review Questions	126
End-of-Chapter Labs	126
Lab 6-1	127
Lab 6-2	127

Part II: Common Ansible Management Tasks

Chapter 7 Using Task Control 131

“Do I Know This Already?” Quiz	131
Foundation Topics	134
Using Loops and Items	134
Working with Loops	134
Using Loops on Variables	135
Using Loops on Multivalued Variables	136
Understanding with_items	137
Using when to Run Tasks Conditionally	139
Working with when	139
Using Conditional Test Statements	141
Testing Multiple Conditions	144
Combining loop and when	145
Combining loop and register	147
Using Handlers	149
Working with Handlers	149
Understanding Handler Execution and Exceptions	154

Dealing with Failures	156
Understanding Task Execution	156
Managing Task Errors	156
Specifying Task Failure Conditions	158
Managing Changed Status	161
Using Blocks	163
Using Blocks with rescue and always Statements	164
Summary	169
Exam Preparation Tasks	169
Review All Key Topics	169
Memory Tables	170
Define Key Terms	170
Review Questions	170
End-of-Chapter Lab	171
Lab 7-1	171
Chapter 8	Deploying Files 173
“Do I Know This Already?” Quiz	173
Foundation Topics	176
Using Modules to Manipulate Files	176
File Module Manipulation Overview	176
Managing File Attributes	176
Managing File Contents	180
Creating and Removing Files	182
Moving Files Around	183
Managing SELinux Properties	187
Managing SELinux File Context	188
Applying Generic SELinux Management Tasks	189
Using Jinja2 Templates	194
Working with Simple Templates	194
Applying Control Structures in Jinja2 Using for	197
Using Conditional Statements with if	198
Using Filters	198
Summary	200

	Exam Preparation Tasks	200
	Review All Key Topics	200
	Memory Tables	201
	Define Key Terms	201
	Review Questions	201
	Exercise Answers	201
	End-of-Chapter Labs	202
	Lab 8-1: Generate an /etc/hosts File	202
	Lab 8-2: Manage a vsftpd Service	202
Chapter 9	Using Ansible Roles	205
	“Do I Know This Already?” Quiz	205
	Foundation Topics	208
	Using Ansible Roles	208
	Understanding Ansible Roles	208
	Understanding Role Location	209
	Using Roles from Playbooks	210
	Creating Custom Roles	210
	Managing Role Dependencies	213
	Understanding File Organization Best Practices	213
	Using Ansible Galaxy Roles	214
	Working with Galaxy	215
	Using the ansible-galaxy Command	216
	Managing Ansible Galaxy Roles	218
	Using RHEL System Roles	219
	Understanding RHEL System Roles	220
	Installing RHEL System Roles	220
	Using the RHEL SELinux System Role	221
	Using the RHEL TimeSync System Role	224
	Summary	225
	Exam Preparation Tasks	225
	Review All Key Topics	226
	Memory Tables	226
	Define Key Terms	226
	Review Questions	226

End-of-Chapter Labs 227

Lab 9-1 227

Lab 9-2 227

Chapter 10 Using Ansible in Large Environments 229

“Do I Know This Already?” Quiz 229

Foundation Topics 232

Advanced Inventory Usage 232

Working with Host Name Patterns 232

Configuring Dynamic Inventory 233

Using the ansible-inventory Command 235

Working with Multiple Inventory Files 236

Optimizing Ansible Processing 237

Managing Parallel Task Execution 237

Managing Serial Task Execution 238

Including and Importing Files 239

Importing Playbooks 240

Importing and Including Task Files 242

Using Variables When Importing and Including Files 243

Summary 247

Exam Preparation Tasks 247

Review All Key Topics 248

Define Key Terms 248

Review Questions 248

End-of-Chapter Lab 249

Lab 10-1 249

Chapter 11 Troubleshooting Ansible 251

“Do I Know this Already?” Quiz 251

Foundation Topics 254

Managing Ansible Errors and Logs 254

Using Check Mode 254

Understanding Output 256

Optimizing Command Output Error Formatting 259

Logging to Files 259

Running Task by Task 260

Using Modules for Troubleshooting and Testing	263
Using the Debug Module	263
Using the uri Module	263
Using the stat Module	265
Using the assert Module	268
Using Tags	271
Troubleshooting Common Scenarios	275
Analyzing Connectivity Issues	275
Analyzing Authentication Issues	276
Summary	277
Exam Preparation Tasks	277
Review All Key Topics	278
Memory Tables	278
Define Key Terms	278
Review Questions	278
End-of-Chapter Lab	279
Lab 11-1	279
Part III: Managing Systems with Ansible	
Chapter 12 Managing Software with Ansible	281
“Do I Know This Already?” Quiz	281
Foundation Topics	284
Using Modules to Manage Packages	284
Configuring Repository Access	284
Managing Software with yum	285
Managing Package Facts	287
Using Modules to Manage Repositories and Subscriptions	289
Setting Up Repositories	290
Managing GPG Keys	291
Managing RHEL Subscriptions	292
Implementing a Playbook to Manage Software	297
Summary	301
Exam Preparation Tasks	302
Review All Key Topics	302

- Memory Tables 302
- Define Key Terms 302
- Review Questions 302
- End-of-Chapter Labs 303
- Lab 12-1 303
- Lab 12-2 303

Chapter 13 Managing Users 305

- “Do I Know This Already?” Quiz 305
- Foundation Topics 308
- Using Ansible Modules to Manage Users and Groups 308
 - Modules Overview 308
 - Managing Users and Groups 308
 - Managing sudo 309
- Managing SSH Connections 313
 - Understanding SSH Connection Management Requirements 313
 - Using the Lookup Plug-in 314
 - Setting Up SSH User Keys 314
- Managing Encrypted Passwords 319
 - Understanding Encrypted Passwords 319
 - Generating Encrypted Passwords 319
 - Using an Alternative Approach 321
- Managing Users Advanced Scenario Exercise 323
- Summary 329
- Exam Preparation Tasks 330
- Review All Key Topics 330
- Memory Tables 330
- Define Key Terms 330
- Review Questions 330
- End-of-Chapter Lab 331
- Lab 13-1 331

Chapter 14 Managing Services and the Boot Process 333

- “Do I Know This Already?” Quiz 333
- Foundation Topics 336

Managing Services	336
Managing Systemd Services	336
Managing cron Jobs	337
Managing at Jobs	338
Managing the Boot Process	341
Managing Systemd Targets	341
Rebooting Managed Hosts	342
Managing the Boot Process and Services Advanced Exercise	344
Summary	346
Exam Preparation Tasks	346
Review All Key Topics	346
Memory Tables	347
Define Key Terms	347
Review Questions	347
End-of-Chapter Lab	348
Lab 14-1	348
Chapter 15 Managing Storage	351
“Do I Know This Already?” Quiz	351
Foundation Topics	354
Discovering Storage-Related Facts	354
Using Storage-Related Facts	354
Using Storage-Related Facts in Conditional Statements	357
Managing Partitions and LVM	364
Creating Partitions	364
Managing Volume Groups and LVM Logical Volumes	366
Creating and Mounting File Systems	367
Configuring Swap Space	368
Configuring Storage Advanced Exercise	371
Summary	379
Exam Preparation Tasks	380
Review All Key Topics	380
Memory Tables	380
Define Key Terms	380

Review Questions 380
End-of-Chapter Lab 381
Lab 15-1 381

Chapter 16 Final Preparation 383

Generic Tips 383
 Verifying Your Readiness 383
 Registering for the Exam 384
 On Exam Day 384
 During the Exam 385
 The Nondisclosure Agreement 386

Part IV: Practice Exams

Practice Exam A 387
Practice Exam B 391

Part V: Appendixes

Appendix A Answers to the “Do I Know This Already?” Quizzes and Review Questions 395
Appendix B Getting Started with Ansible Tower 415
Appendix C *Red Hat RHCE 8 (EX294) Cert Guide* Exam Updates 431
 Glossary 433
 Index 439

Online Elements:

 Practice Exam C
 Practice Exam D
Appendix D Memory Tables
Appendix E Memory Tables Answer Key
Appendix F Study Planner
 Glossary

About the Author

Sander van Vugt has been teaching Linux classes since 1995 and has written more than 60 books about different Linux-related topics, including the best-selling *RHCSA-RHCE 7 Cert Guide* and the *RHCSA 8 Cert Guide: EX200*. Sander is also the author of more than 25 video courses, including his RHCSA and RHCE Complete Video Courses, *Hands-On Ansible LiveLessons*, and many other titles. He teaches courses for customers around the world and is also a regular speaker at major conferences related to open-source software. Sander is also the founder of the Living Open Source Foundation, a nonprofit organization that teaches open-source courses in African countries.

Dedication

This book is dedicated to my family: Florence, Franck, and Alex. Together we've made great accomplishments over the past year.

Acknowledgments

This book could not have been written without the help of all the people who contributed to it. To start, I want to thank the people at Pearson—Denise Lincoln and Ellie Bru in particular. We’ve worked a lot together over the past years, and this book is another milestone on our road to success! It has been fantastic how you both have helped me to realize this book in just two months!

Next, I want to thank my technical reviewers. Big thanks to Bo and John! Thanks to your great feedback, I’ve been able to apply important improvements to the contents of this book. Also, a special thanks to Ettiene Esterhuizen from New Zealand and Santos Venter Chibenga and Robert Charles Muchendu from the African Living Open Source Community, who helped me as volunteer reviewers. And last but not least, thanks to my fellow instructor and colleague Pascal van Dam, who helped me make some important last-minute improvements.

About the Technical Reviewers

John McDonough has more than 30 years of development experience; currently, John is a developer advocate for Cisco DevNet. As a developer advocate, John writes code and creates DevNet Learning Labs about how to write code; writes blogs about writing code; and presents at Cisco Live, SXSW, AnsibleFest, and other industry events. John focuses on the Cisco computing systems products, Cisco UCS, and Cisco Intersight. John's career at Cisco has varied from product engineer to custom application developer, technical marketing engineer, and now a developer advocate.

William “Bo” Rothwell crossed paths with a TRS-80 Micro Computer System (affectionately known as a “Trash 80”) at the impressionable age of 14. Soon after, the adults responsible for Bo made the mistake of leaving him alone with the TRS-80. He immediately dismantled it and held his first computer class, showing his friends what made this “computer thing” work.

Since this experience, Bo's passion for understanding how computers work and sharing this knowledge with others has resulted in a rewarding career in IT training. His experience includes Linux, UNIX, and programming languages such as Perl, Python, Tcl, and BASH. He is the founder and president of One Course Source, an IT training organization.

We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

We welcome your comments. You can email to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book.

When you write, please be sure to include this book's title and author as well as your name and email address. We will carefully review your comments and share them with the author and editors who worked on the book.

Email: community@informit.com

Introduction

Welcome to the *Red Hat RHCE 8 (EX294) Cert Guide*! With the release of Red Hat Enterprise Linux 8, Red Hat has decided to take a completely new direction for the RHCE exam. The exam is now completely about managing configurations with Ansible. This is a great choice because in the current IT landscape the days of the system administrator who applies specialized skills to tune individual servers is over. Today the work is all about automation, and Ansible has rapidly become one of the most important solutions to do so.

As a Linux instructor with more than 25 years of experience, I have been certified for both the RHCSA and RHCE exams for every RHEL version since RHEL 4. Taking the exams myself has helped me keep current on the progression of the exam, what is new, and what is different. I am thrilled to be able to share my knowledge with you in this comprehensive Cert Guide so you can get the guidance you need to pass your RHCE RHEL 8 EX294 exam.

As you will see, this Cert Guide covers every objective in the updated RHCE exam, with 16 chapters, more than 40 exercises, 4 practice exams (2 printed in the book and 2 on the companion website), and 1 hour of video training. This *Red Hat RHCE 8 (EX294) Cert Guide* is the best resource you can get to prepare for and pass the exams.

Goals and Methods

To learn the topics described in this book, I recommend that you create your own testing environment, which is explained in Chapter 2, “Installing Ansible.” You cannot become an RHCE without practicing a lot. To get familiar with the topics in the chapters, here is what I recommend:

- Read the explanation in the chapters and study the code examples that are provided in the listings. For your convenience, the listings are also provided in the book GitHub repository at <https://github.com/sandervanvugt/rhce8-book>. Study the examples and try to understand what they do.
- Walk through all of the numbered exercises in the book. The numbered exercises provide step-by-step instructions, and you should follow along with all of them, to walk through configuration tasks and learn how to manage specific features.
- At the end of each chapter, there’s an end-of-chapter lab. This lab is much like the lab assignments that you will find on the exam.

Within the exercises included in every chapter of the book, you will find all the examples you need to understand what is on the exam and thoroughly learn the material needed to pass it. The exercises in the chapters provide step-by-step procedure descriptions that you can work through to find working solutions so that you can get real experience before taking the tests. Although you may feel familiar with some topics, it's a good idea to work through all of the exercises in the book. The RHCE exam is hands-on, which can be a lot of pressure on test day. The exercises in each chapter help provide the practice you need to make sure you have the experience you need to not make small errors and mistakes while taking the exam. The exercises are the best way to make sure you work through common errors and learn from your mistakes before you take the test.

Each chapter also includes an end-of-chapter lab. These labs ask questions that are similar to the questions that you might encounter on the exam so you can use them to practice. I have purposely excluded solutions for these labs for a few reasons: (1) you need to train yourself to verify your work before test day because you will be expected to do this on the exam; (2) while taking the test, you will be required to verify for yourself whether your solution is working as expected; and (3) most labs have multiple solutions and I don't want to suggest that my solution is the right one and yours is wrong because it takes a different approach. Your solution is as good as mine, as long as it accomplishes what was asked for in the exercise.

Other Resources

This book contains everything you need to pass the exam, but if you want more guidance and practice, I have a number of video training titles available to help you study, including the following:

- *Red Hat Certified Engineer (RHCE) 3/ed Complete Video Course*
- *Hands-on Ansible LiveLessons*

Apart from these products, you might also appreciate my website: rhatcert.com. Through this website, I provide updates on anything that is useful to exam candidates. I recommend that you register on the website so that I can send you messages about important updates that I've made available. Also, you'll find occasional video updates on my YouTube channel: [rhatcert](https://www.youtube.com/channel/UCrhatcert). I hope that all these resources provide you with everything you need to pass the Red Hat exams in an affordable way! Good luck!

Who Should Read This Book?

This book is written as an RHCE exam preparation guide. That means that you should read it if you want to increase your chances of passing the RHCE exam.

I have also written this book to help you become familiar with Ansible. So even if you're not interested in the RHCE EX294 exam at all, this book will teach you everything you need to know to get your Ansible career up and running.

So, why should you consider passing the RHCE exam? That question is simple to answer. Linux has become a very important operating system, and qualified professionals are sought after all over the world. If you want to work as a Linux professional and prove your skills, the RHCE certificate really helps. Having these certificates dramatically increases your chances of becoming hired as a Linux professional. Notice that in order to get RHCE certified, you must hold a current RHCSA certification. You can take the RHCE EX294 exam before you are RHCSA certified, but you can call yourself an RHCE only if you have passed both the RHCSA exam and the RHCE exam.

How This Book Is Organized

This book is organized as a reference guide to help you prepare for the exams. If you're new to the topics, you can just read it cover to cover. You can also read the individual chapters that you need to fine-tune your skills in this book. Every chapter starts with a “Do I Know This Already?” quiz. This quiz asks questions about 10 topics that are covered in each chapter and provides a simple tool to check whether you're already familiar with the topics covered in a chapter. These quizzes do not represent the types of questions you will get on the real exam though.

The best exam preparation is offered in the RHCE practice exams; these are an essential part of readying yourself for the real testing experience. You might be able to provide the right answer to the multiple-choice chapter questions, but that doesn't mean that you can create the configurations when you take the tests. We have included two practice exams in the printed book. The book's companion website then includes two additional practice exams as well as flashcards created from the book's glossary so you can further test your knowledge and skills. You will also find one hour of video from my *Red Hat Certified Engineer (RHCE) 3/ed Complete Video Course*.

The following topics are covered in the chapters:

- **Chapter 1, “Understanding Configuration Management”:** In this chapter, you learn about Ansible as a solution. The chapter explains what can be done with Ansible and how Ansible relates to other solutions for configuration management.
- **Chapter 2, “Installing Ansible”:** This chapter covers installation of Ansible. You learn what is needed to set up the Ansible control node, as well as the other parts of the Ansible software.

- **Chapter 3, “Setting Up an Ansible Managed Environment”:** In this chapter you learn how to get started with node management. The chapter explains what is needed on the managed nodes as well as the essential Ansible configuration files that are required to reach out to the managed nodes.
- **Chapter 4, “Using Ad Hoc Commands”:** In this chapter you learn about Ansible modules. Modules are the heart of Ansible; they provide solutions for everything that Ansible can do, and the easiest way to use these modules is in ad hoc commands. In this chapter you learn how to work with them.
- **Chapter 5, “Getting Started with Playbooks”:** This chapter provides an introduction to working with playbooks. You learn about YAML, the language used to write playbooks, and how to structure a playbook using plays and tasks.
- **Chapter 6, “Working with Variables and Facts”:** In Ansible, variables can be used to provide dynamic values to specific configuration items. Using variables enables you to separate the static code in a playbook with host-specific information. In this chapter you learn how to work with variables as well as Ansible facts, which are variables that are automatically set for managed nodes.
- **Chapter 7, “Using Task Control”:** To make Ansible smart, you must apply task control. Using task control enables you to run tasks conditionally, and that can be done in many ways. You learn how to use tests, to test for a specific condition, as well as loops that allow you to evaluate a range of items, and handlers, which allow for task execution only if another task was executed successfully.
- **Chapter 8, “Deploying Files”:** Ansible is used for configuration management, and configuration on Linux is stored in files. Hence, managing files is a key skill in Ansible. In this chapter you learn how to use modules to modify files and how to use templates to automatically set up configuration files with specific parameters obtained from facts or variables.
- **Chapter 9, “Using Ansible Roles”:** When you are working with Ansible, it’s good if code can be reused. That is what Ansible roles are all about. In this chapter you learn how to work with roles, which are provided through Ansible Galaxy, or as RHEL system roles.
- **Chapter 10, “Using Ansible in Large Environments”:** When working with Ansible in large environments, you should know about a few specific techniques. These techniques are covered in this chapter. You learn how to optimize Ansible by modifying the number of concurrent tasks that can be executed. You also learn how to work with includes and imports, which allow you to set up modular playbooks.

- **Chapter 11, “Troubleshooting Ansible”:** In some cases your playbook might not give you the desired result. Then you need to start troubleshooting. This chapter contains not only all you need to know about troubleshooting, including some best practices while developing playbooks, but also information about modules that can be used to make troubleshooting easier.
- **Chapter 12, “Managing Software with Ansible”:** This is the first chapter about specific common tasks that you can perform with Ansible. In this chapter you learn how to set up repositories and how to manage software packages with Ansible.
- **Chapter 13, “Managing Users”:** To do anything on Linux, you need user accounts. In this chapter you learn all that is needed to create user accounts, including setting encrypted passwords.
- **Chapter 14, “Managing Services and the Boot Process”:** Occasionally, you might want to run scheduled jobs. These jobs will be executed at a specific time, using either cron or at. In this chapter you learn how to do that, and you also learn how to manage the systemd default target.
- **Chapter 15, “Managing Storage”:** Setting up storage is a key task when working with Linux. In this chapter you learn how to automate storage configuration with Ansible. You also learn how to discover disk devices available on your managed systems and how to set them up, using partitions, logical volumes, filesystems, and mounts.
- **Chapter 16, “Final Preparation”:** In this chapter you get some final exam preparation tasks. It contains some test exams and many tips that help you maximize your chances of passing the exam.

How to Use This Book

To help you customize your study time using these books, the core chapters have several features that help you make the best use of your time:

- **“Do I Know This Already?” Quizzes:** Each chapter begins with a quiz that helps you determine the amount of time you need to spend studying that chapter.
- **Foundation Topics:** These are the core sections of each chapter. They explain the protocols, concepts, and configuration for the topics in that chapter.
- **Exam Preparation Tasks:** At the end of the “Foundation Topics” section of each chapter, the “Exam Preparation Tasks” section lists a series of study activities that should be done at the end of the chapter. Each chapter includes the

activities that make the most sense for studying the topics in that chapter. The activities include the following:

- **Review Key Topics:** The Key Topic icon is shown next to the most important items in the “Foundation Topics” section of the chapter. The Key Topics Review activity lists the key topics from the chapter and their corresponding page numbers. Although the contents of the entire chapter could be on the exam, you should definitely know the information listed in each key topic.
- **Complete Tables and Lists from Memory:** To help you exercise your memory and memorize some lists of facts, many of the more important lists and tables from the chapter are included in a document on the DVD and companion website. This document lists only partial information, allowing you to complete the table or list.
- **Define Key Terms:** This section lists the most important terms from the chapter, asking you to write a short definition and compare your answer to the glossary at the end of this book.
- **Review Questions:** Questions at the end of each chapter measure insight in the topics that were discussed in the chapter.
- **End-of-Chapter Labs:** These real labs give you the right impression on what an exam assignment looks like. The end-of-chapter labs are your first step in finding out what the exam tasks really look like.

Other Features

In addition to the features in each of the core chapters, this book, as a whole, has additional study resources on the companion website, including the following:

- **Four practice exams:** The companion website contains the four practice exams: two provided in the book and two available on the companion website.
- **Flashcards:** The companion website contains interactive flashcards created from the glossary terms in the book so you can better learn key terms and test your knowledge.
- **More than one hour of video training:** The companion website contains more than one hour of video training from the best-selling *Red Hat Certified Engineer (RHCE) 3/ed Complete Video Course*.

Book Organization, Chapters, and Appendixes

I have also included a table that details where every objective in the RHCE exam is covered in this book so that you can more easily create a successful plan for passing the tests.

Table 1 RHCE Objectives

Objective	Chapter Title	Chapter	Page
Understand core components of Ansible: Inventories	Setting Up an Ansible Managed Environment	3	31
Understand core components of Ansible: Modules	Using Ad Hoc Commands	4	47
Understand core components of Ansible: Variables	Working with Variables and Facts	6	97
Understand core components of Ansible: Facts	Working with Variables and Facts	6	97
Understand core components of Ansible: Plays	Getting Started with Playbooks	5	69
Understand core components of Ansible: Playbooks	Getting Started with Playbooks	5	69
Understand core components of Ansible: Configuration files	Setting Up an Ansible Managed Environment	3	31
Understand core components of Ansible: Use provided documentation	Using Ad Hoc Commands	4	47
Install and configure an Ansible control node: Install required packages	Installing Ansible	2	15
Install and configure an Ansible control node: Create a static host inventory file	Setting Up an Ansible Managed Environment	3	31
Install and configure an Ansible control node: Create a configuration file	Setting Up an Ansible Managed Environment	3	31
Install and configure an Ansible control node: Create and use static inventories	Setting Up an Ansible Managed Environment	3	31
Install and configure an Ansible control node: Manage parallelism	Using Ansible in Large Environments	10	229
Configure Ansible managed nodes: Create and distribute SSH keys to managed nodes	Installing Ansible	2	15
Configure Ansible managed nodes: Configure privilege escalation on managed nodes	Installing Ansible	2	15

Objective	Chapter Title	Chapter	Page
Configure Ansible managed nodes: Validate a working configuration using ad hoc Ansible commands	Using Ad Hoc Commands	4	47
Script administration tasks: Create simple shell scripts	Using Ad Hoc Commands	4	47
Script administration tasks: Create simple shell scripts that run ad hoc Ansible commands	Using Ad Hoc Commands	4	47
Create Ansible plays and playbooks: Know how to work with commonly used Ansible modules	Using Ad Hoc Commands	4	47
Create Ansible plays and playbooks: Use variables to retrieve the results of running a command	Working with Variables and Facts	6	97
Create Ansible plays and playbooks: Use conditionals to control play execution	Using Task Control	7	131
Create Ansible plays and playbooks: Configure error handling	Using Task Control	7	131
Create Ansible plays and playbooks: Create playbooks to configure systems to a specified state	Getting Started with Playbooks	5	69
Use Ansible modules for system administration tasks that work with: Software packages and repositories	Managing Software with Ansible	12	281
Use Ansible modules for system administration tasks that work with: Services	Managing Processes and Tasks	14	333
Use Ansible modules for system administration tasks that work with: Firewall rules	Getting Started with Playbooks	5	69
Use Ansible modules for system administration tasks that work with: File systems	Managing Storage	15	351
Use Ansible modules for system administration tasks that work with: Storage devices	Managing Storage	15	351
Use Ansible modules for system administration tasks that work with: File content	Deploying Files	8	173
Use Ansible modules for system administration tasks that work with: Archiving	Managing Storage	15	351

Objective	Chapter Title	Chapter	Page
Use Ansible modules for system administration tasks that work with: Scheduled tasks	Managing Processes and Tasks	14	333
Use Ansible modules for system administration tasks that work with: Security	Managing Users	13	305
Use Ansible modules for system administration tasks that work with: Users and Groups	Managing Users	13	305
Work with roles: Create roles	Using Ansible Roles	9	205
Work with roles: Download roles from an Ansible Galaxy and use them	Using Ansible Roles	9	205
Use advanced Ansible features: Create and use templates to create customized configuration files	Deploying Files	8	173
Use advanced Ansible features: Use Ansible Vault in playbooks to protect sensitive data	Working with Variables and Facts	6	97

Where Are the Companion Content Files?

Register this print version of *Red Hat RHCE 8 (EX294) Cert Guide* to access the bonus content online.

This print version of this title comes with companion content. You have online access to these files by following these steps:

1. Go to www.pearsonITcertification.com/register and log in or create a new account.
2. Enter the ISBN: **9780136872436**.
3. Answer the challenge question as proof of purchase.
4. Click on the **Access Bonus Content** link in the Registered Products section of your account page to be taken to the page where your downloadable content is available.

Please note that many of our companion content files can be very large, especially image and video files.

If you are unable to locate the files for this title by following the steps, please visit www.pearsonITcertification.com/contact and select the Site Problems/Comments option. Our customer service representatives will assist you.

Credits

Chapter opener images by Charlie Edwards/Photodisc/Getty Images

Chapter 1 quote, “a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality,” © Len Bass, Ingo Weber, and Liming Zhu, *DevOps: A Software Architect’s Perspective*, Boston, MA: Addison-Wesley Professional, 2015.

Chapter 4 quote, “Your work will be evaluated by applying the playbooks created during the exam against freshly installed systems and verifying that those systems and services work as specified,” © 2020 Red Hat, Inc.

Cover image: Branislav Nenin/Shutterstock

Deploying Files

The following RHCE exam objectives are covered in this chapter:

- Use Ansible modules for system administration tasks that work with:
 - File contents
- Use advanced Ansible features
 - Create and use templates to create customized configuration files

“Do I Know This Already?” Quiz

The “Do I Know This Already?” quiz allows you to assess whether you should read this entire chapter thoroughly or jump to the “Exam Preparation Tasks” section. If you are in doubt about your answers to these questions or your own assessment of your knowledge of the topics, read the entire chapter. Table 8-1 lists the major headings in this chapter and their corresponding “Do I Know This Already?” quiz questions. You can find the answers in Appendix A, “Answers to the ‘Do I Know This Already?’ Quizzes and Review Questions.”

Table 8-1 “Do I Know This Already?” Section-to-Question Mapping

Foundation Topics Section	Questions
Using Modules to Manipulate Files	1–6
Managing SELinux Properties	7, 8
Using Jinja2 Templates	9, 10

1. Which module should you use to check the current permission mode on a file?
 - a. stat
 - b. file
 - c. permissions
 - d. acl

2. Which module should you use to replace a line of text in a configuration file with another line of text?
 - a. copy
 - b. regex
 - c. lineinfile
 - d. blockinfile

3. Which of the following shows correct syntax for a **when** statement that runs a task only if the permission mode as discovered by the stat module and registered to the st variable is not set to 0640?
 - a. `st.mode != '0640'`
 - b. `st.stat.mode != 0640`
 - c. `st.stat.mode != '0640'`
 - d. `st.mode != 0640`

4. Which of the following lines shows correct use of the lineinfile module to find a line that begins with PermitRootLogin based on a regular expression?
 - a. `line: "PermitRootLogin"`
 - b. `line: "^PermitRootLogin"`
 - c. `regexp: "PermitRootLogin"`
 - d. `regexp: "^PermitRootLogin"`

5. Which of the following is not a common task that the file module can do?
 - a. Remove files
 - b. Copy a line of text into a file
 - c. Create links
 - d. Set permissions

6. Which module can you use to copy a file from a managed node to the control node?
 - a. copy
 - b. file
 - c. sync
 - d. fetch

7. Different modules can be used when working with SELinux. Which of the following modules should you avoid?
 - a. file
 - b. sefcontext
 - c. command
 - d. selinux

8. After you set an SELinux context, the Linux **restorecon** command must be executed. How would you do this?
 - a. Use the command module to run the **restorecon** command.
 - b. Use the restorecon module.
 - c. Use the selinux module.
 - d. No further action is needed; this is done automatically when using the appropriate SELinux module.

9. What do you need to transform the contents of a variable to the JSON format?
 - a. The lineinfile module
 - b. A Jinja2 template
 - c. A filter
 - d. The copy module

10. What should you use to process host-specific facts from a template?
 - a. The hostvars macro
 - b. The hostvars magic variable
 - c. The hostvars module
 - d. The hostvars filter

Foundation Topics

Using Modules to Manipulate Files

Managing files is an important task for Linux administrators. Different types of manipulations are performed on files on a frequent basis. They include managing files, managing file contents, and moving files around. In this section you learn how to use Ansible modules to apply these different tasks.

File Module Manipulation Overview

Many modules are available to manage different aspects of files. Table 8-2 provides an overview of some of the most commonly used file modules.

Table 8-2 File Manipulation Module Overview

Module	Use
copy	Copies files to remote locations
fetch	Fetches files from remote locations
file	Manages files and file properties
acl	Works with file system ACLs
find	Finds files based on any property
lineinfile	Manages lines in text files
blockinfile	Manages blocks in text files
replace	Replaces strings in text files based on regex
synchronize	Performs rsync-based synchronization tasks
stat	Retrieves file or file system status

Most of these modules are discussed in the following sections. When using file-related modules, you might need a module that is not discussed here. If that is the case, the best approach is to use the **ansible-doc** command. When you use this command on any module, you always see related modules mentioned in the SEE ALSO section of the documentation.

Managing File Attributes

If you need to work with file attributes, the stat module and the file module come in handy. The stat module enables you to retrieve file status information. Because this module gets status information and is not used to change anything, you mainly use it

to check specific file properties and perform an action if the properties are not set as expected. In Listing 8-1 you can see a playbook that uses the `stat` and `debug` modules to explore what exactly the `stat` module is doing. Listing 8-2 shows the output shown while running `ansible-playbook listing81.yaml`.

Listing 8-1 Exploring the `stat` Module

```
---
- name: stat module tests
  hosts: ansible1
  tasks:
  - stat:
    path: /etc/hosts
    register: st
  - name: show current values
    debug:
      msg: current value of the st variable is {{ st }}
```

Listing 8-2 Running `ansible-playbook listing81.yaml`

```
[ansible@control ~]$ ansible-playbook listing81.yaml

PLAY [stat module tests] *****

TASK [Gathering Facts] *****
ok: [ansible1]

TASK [stat] *****
ok: [ansible1]

TASK [show current values] *****
ok: [ansible1] => {
  "msg": "current value of the st variable is {'changed':
False, 'stat': {'exists': True, 'path': '/etc/hosts', 'mode':
'0644', 'isdir': False, 'ischr': False, 'isblk': False, 'isreg':
True, 'isfifo': False, 'islnk': False, 'issock': False, 'uid': 0,
'gid': 0, 'size': 158, 'inode': 16801440, 'dev': 64768, 'nlink':
1, 'atime': 1586230060.147566, 'mtime': 1536580263.0, 'ctime':
1584958718.8117938, 'wusr': True, 'rusr': True, 'xusr': False, 'wgrp':
False, 'rgrp': True, 'xgrp': False, 'woth': False, 'roth': True,
'xoth': False, 'devuid': False, 'isgid': False, 'blocks': 8, 'block_
size': 4096, 'device_type': 0, 'readable': True, 'writeable': True,
'executable': False, 'pw_name': 'root', 'gr_name': 'root', 'checksum':
'7335999eb54c15c67566186bdfc46f64e0d5alaa', 'mimetype': 'text/plain',
'charset': 'us-ascii', 'version': '408552077', 'attributes': [],
'attr_flags': ''}, 'failed': False}"
}
```

```
PLAY RECAP *****
ansible1                : ok=3    changed=0    unreachable=0
failed=0    skipped=0    rescued=0    ignored=0
```

As you can see from Listing 8-2, the `stat` module returns many file properties. It tests which permission mode is set, whether it is a link, which checksum is set on the file, and much more. For a complete list of output data, you can consult the documentation as provided while running **ansible-doc stat**.

Based on the output that is provided, a conditional test can be performed. The sample playbook in Listing 8-3 shows how this can be done and how the playbook can write a message if the expected permissions mode is not set.

Listing 8-3 Performing File State Tests with the `stat` Module

```
---
- name: stat module tests
  hosts: ansible1
  tasks:
  - command: touch /tmp/statfile
  - stat:
    path: /tmp/statfile
    register: st
  - name: show current values
    debug:
      msg: current value of the st variable is {{ st }}
  - fail:
    msg: "unexpected file mode, should be set to 0640"
    when: st.stat.mode != '0640'
```

As you can see in the playbook output in Listing 8-4, the playbook fails with the unexpected file mode message. Also notice the warning in the Listing 8-4 output: it tells you that there is a better solution to do what you wanted to do here. This happens on multiple occasions when you might have selected a module that is not the best solution for the task you want to perform. Remember: Using the `command` module will work in almost all cases, but often a better solution is available.

Listing 8-4 Running **ansible-playbook listing83.yaml** Result

```
[ansible@control ~]$ ansible-playbook listing83.yaml

PLAY [stat module tests] *****
```

```

TASK [Gathering Facts] *****
ok: [ansible1]

TASK [command] *****
[WARNING]: Consider using the file module with state=touch rather
than running 'touch'. If you need to use command because file is
insufficient you can add 'warn: false' to this command task or set
'command_warnings=False' in ansible.cfg to get rid of this message.
changed: [ansible1]

TASK [stat] *****
ok: [ansible1]

TASK [show current values] *****
ok: [ansible1] => {
    "msg": "current value of the st variable is {'changed':
False, 'stat': {'exists': True, 'path': '/tmp/statfile', 'mode':
'0644', 'isdir': False, 'ischr': False, 'isblk': False, 'isreg':
True, 'isfifo': False, 'islnk': False, 'issock': False, 'uid': 0,
'gid': 0, 'size': 0, 'inode': 51440456, 'dev': 64768, 'nlink': 1,
'atime': 1586253087.057596, 'mtime': 1586253087.057596, 'ctime':
1586253087.057596, 'wusr': True, 'rusr': True, 'xusr': False, 'wgrp':
False, 'rgrp': True, 'xgrp': False, 'woth': False, 'roth': True,
'xoth': False, 'isuid': False, 'isgid': False, 'blocks': 0, 'block_
size': 4096, 'device_type': 0, 'readable': True, 'writeable': True,
'executable': False, 'pw_name': 'root', 'gr_name': 'root', 'checksum':
'da39a3ee5e6b4b0d3255bfef95601890afd80709', 'mimetype': 'inode/x-
empty', 'charset': 'binary', 'version': '158303785', 'attributes': [],
'attr_flags': ''}, 'failed': False}"
}

TASK [fail] *****
fatal: [ansible1]: FAILED! => {"changed": false, "msg": "unexpected
file mode, should be set to 0640"}

PLAY RECAP *****
ansible1          : ok=4      changed=1    unreachable=0
failed=1         skipped=0    rescued=0    ignored=0

```

In the earlier examples in this section, you saw how you can use the stat module to show different types of file properties. Based on the output of the stat module, you may use the file module to set specific file properties. In Listing 8-5 you can see how the playbook from Listing 8-3 is rewritten to automatically set the desired permissions state.

Listing 8-5 Using the file Module to Correct File Properties Discovered with stat

```
---
- name: stat module tests
  hosts: ansible1
  tasks:
  - command: touch /tmp/statfile
  - stat:
    path: /tmp/statfile
    register: st
  - name: show current values
    debug:
      msg: current value of the st variable is {{ st }}
  - name: changing file permissions if that's needed
    file:
      path: /tmp/statfile
      mode: 0640
      when: st.stat.mode != '0640'
```

EXAM TIP In the examples in this chapter, some tasks don't have a name assigned. Using a name for each task is not required; however, it does make troubleshooting a lot easier if each task does have a name. For that reason, on the exam it's a good idea to use names anyway. Doing so makes it easier to identify which tasks lead to which specific result.

Managing File Contents

If you need to manage file contents, multiple modules can be useful. The `find` module enables you to find files, just like the Linux `find` command. The `lineinfile` module enables you to manipulate lines in files, and `blockinfile` enables you to manipulate complete blocks of text. Also don't forget the `copy` module. We look at it in the next section, but you can also use it to copy a specified text to a file. For managing text operations on files, however, it is recommended that you use `lineinfile` or `blockinfile` instead because these give more options to specify where exactly the text should be written to.

Listing 8-6 shows an example where `lineinfile` is used to change a string, based on a regular expression.

Listing 8-6 Changing File Contents Using lineinfile

```
---
- name: configuring SSH
  hosts: all
  tasks:
    - name: disable root SSH login
      lineinfile:
        dest: /etc/ssh/sshd_config
        regexp: "^PermitRootLogin"
        line: "PermitRootLogin no"
        notify: restart sshd

  handlers:
    - name: restart sshd
      service:
        name: sshd
        state: restarted
```

As you can see in Listing 8-6, lineinfile uses the **dest** key to specify the filename. Next, a regular expression is used to search for lines that have text starting with **PermitRootLogin**. If this regular expression is found, it is changed into the line **PermitRootLogin no**.

You can use the lineinfile module to manipulate a single line in a file. In some cases you have to manage multiple lines in a file. In that case, you can use the blockinfile module. Listing 8-7 provides an example.

Listing 8-7 Using blockinfile to Manipulate Multiple Lines of Text

```
---
- name: modifying file
  hosts: all
  tasks:
    - name: ensure /tmp/hosts exists
      file:
        path: /tmp/hosts
        state: touch
    - name: add some lines to /tmp/hosts
      blockinfile:
        path: /tmp/hosts
        block: |
          192.168.4.110 host1.example.com
          192.168.4.120 host2.example.com
        state: present
```

Based on what you've learned so far, the use of `blockinfile` should be easy to understand. Just remember the use of the `|` after **block:**. This character is used to specify that the next couple of lines should be treated as lines, adding the newline character to the end of the line. Alternatively, you could use **block: >**, but that would add one long line to the destination file.

Notice that when `blockinfile` is used, the text specified in the block is copied with a start and end indicator. See Listing 8-8 for an example:

Listing 8-8 Resulting File Modification by `blockinfile`

```
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.
localhost4
::1 localhost localhost.localdomain localhost6 localhost6.
localhost6
192.168.4.200. control.example.com. control
192.168.4.201 ansible1.example.com ansible1
192.168.4.202 ansible2.example.com ansible2

# BEGIN ANSIBLE MANAGED BLOCK
192.168.4.110 host1.example.com
192.168.4.120 host2.example.com
# END ANSIBLE MANAGED BLOCK
```

Creating and Removing Files

In an earlier example in this chapter you saw how the `command` module was used to create a new file by using the Linux **touch** command. While running this playbook, you saw a warning that you shouldn't do it this way, but you should use the `file` module instead, and that is totally right.

You can use the `file` module to perform some pretty common tasks:

Key Topic

- Create new files or directories
- Create links
- Remove files
- Set permissions and ownership

Listing 8-9 shows a sample playbook where the `file` module is used to create a new directory and in that directory create an empty file, after which the same `file` module is used again to remove the directory recursively. This approach is not very useful, but at least it shows you some of the most common uses of the `file` module.

Listing 8-9 Creating and Removing Files with the file Module

```
---
- name: using the file module
  hosts: ansible1
  tasks:
  - name: create directory
    file:
      path: /newdir
      owner: ansible
      group: ansible
      mode: 770
      state: directory
  - name: create file in that directory
    file:
      path: /newdir/newfile
      state: touch
  - name: show the new file
    stat:
      path: /newdir/newfile
      register: result
  - debug:
      msg: |
          This shows that newfile was created
          "{{ result }}"
  - name: removing everything again
    file:
      path: /newdir
      state: absent
```

In Listing 8-9, you can see that the last task is configured to remove a directory. Just specifying the path to the directory and **state: absent** recursively removes the directory. You don't need to specify any other options here, and the **recurse** key also is not required.

Moving Files Around

Three modules are particularly useful for moving files around. The copy module copies a file from the Ansible control host to a managed machine. The fetch module enables you to do the opposite, and the synchronize module performs Linux rsync-like tasks, ensuring that a file from the control host is synchronized to a file with that name on the managed host. The main difference between copy and synchronize

is that the copy module always creates a new file, whereas the synchronize module updates a current existing file. In Listing 8-10 you can see how these modules are used.

Listing 8-10 Moving a File Around with Ansible

```
---
- name: file copy modules
  hosts: all
  tasks:
  - name: copy file demo
    copy:
      src: /etc/hosts
      dest: /tmp/
  - name: add some lines to /tmp/hosts
    blockinfile:
      path: /tmp/hosts
      block: |
        192.168.4.110 host1.example.com
        192.168.4.120 host2.example.com
      state: present
  - name: verify file checksum
    stat:
      path: /tmp/hosts
      checksum_algorithm: md5
    register: result
  - debug:
      msg: "The checksum of /tmp/hosts is {{ result.stat.checksum }}"
  - name: fetch a file
    fetch:
      src: /tmp/hosts
      dest: /tmp/
```

After running the playbook in Listing 8-10, you might expect to find the file `/tmp/hosts` on the Ansible control machine. This, however, is not the case, and the reason is easy to understand. Ansible playbooks typically are used on multiple hosts, so if a file is fetched from a managed host, it must be stored in a unique location. To guarantee the uniqueness, Ansible creates a subdirectory for each managed host in the `dest` directory and puts the file that fetch has copied from the remote host in that subdirectory. So the result of the playbook in Listing 8-10 is stored as `/tmp/ansible1/hosts` and `/tmp/ansible2/hosts`. You practice working with files in Exercise 8-1.

Exercise 8-1 Managing Files with Ansible

1. Create a file with the name `exercise81.yaml` and give it the following play header:

```
---
- name: testing file manipulation skills
  hosts: ansible1
  tasks:
```

2. Add a task that creates a new empty file:

```
---
- name: testing file manipulation skills
  hosts: ansible1
  tasks:
  - name: create a new file
    file:
      name: /tmp/newfile
      state: touch
```

3. Use the `stat` module to check on the status of the new file:

```
---
- name: testing file manipulation skills
  hosts: ansible1
  tasks:
  - name: create a new file
    file:
      name: /tmp/newfile
      state: touch
  - name: check status of the new file
    stat:
      path: /tmp/newfile
      register: newfile
```

4. To see what the `stat` module is doing, add a line that uses the `debug` module:

```
- name: testing file manipulation skills
  hosts: ansible1
  tasks:
  - name: create a new file
    file:
      name: /tmp/newfile
      state: touch
  - name: check status of the new file
    stat:
      path: /tmp/newfile
```

```

    register: newfile
  - name: for debugging only
    debug:
      msg: the current values for newfile are {{ newfile }}

```

5. Now that you understand which values are stored in `newfile`, you can add a conditional playbook that changes the current owner if not set correctly:

```

---
- name: testing file manipulation skills
  hosts: ansible1
  tasks:
  - name: create a new file
    file:
      name: /tmp/newfile
      state: touch
  - name: check status of the new file
    stat:
      path: /tmp/newfile
    register: newfile
  - name: for debugging only
    debug:
      msg: the current values for newfile are {{ newfile }}
  - name: change file owner if needed
    file:
      path: /tmp/newfile
      owner: ansible
    when: newfile.stat.pw_name != 'ansible'

```

6. Add a second play to the playbook that fetches a remote file:

```

- name: fetching a remote file
  hosts: ansible1
  tasks:
  - name: fetch file from remote machine
    fetch:
      src: /etc/motd
      dest: /tmp

```

7. Now that you have fetched the file so that it is on the Ansible control machine, use `blockinfile` to edit it:

```

- name: adding text to the file that is now on localhost
  hosts: localhost
  tasks:
  - name: add a message
    blockinfile:

```

```

path: /tmp/ansible1/etc/motd
block: |
    welcome to this server
    for authorized users only
state: present

```

8. In the final step, copy the modified file to `ansible2` by including the following play:

```

- name: copy the modified file to ansible2
  hosts: ansible2
  tasks:
  - name: copy motd file
    copy:
      src: /tmp/ansible1/etc/motd
      dest: /tmp

```

9. At this point you're ready to run the playbook. Type **ansible-playbook exercise81.yaml** to run it and observe the results.

10. Type **ansible ansible2 -a "cat /tmp/motd"** to verify that the modified motd file was successfully copied to `ansible2`.

Managing SELinux Properties

In the security of any Linux system, SELinux is an important component. SELinux can be used on files to manage file context; apart from that, context can be set on ports; and SELinux properties can be managed using Booleans. Ansible has a few modules that allow for making changes to the SELinux configuration, which are listed in Table 8-3.

TIP To work with SELinux in Ansible, you need to have knowledge about SELinux. This is a part of the RHCSA-level knowledge that is required for anyone who wants to take EX294. This section does not explain SELinux itself. For more information about SELinux, consult the Red Hat RHCSA 8 Cert Guide.

Table 8-3 Modules for Managing Changes on SELinux

Module	Use
file	Manages context on files but not in the SELinux policy
sefcontext	Manages file context in the SELinux policy
command	Is required to run the restorecon command after using <code>sefcontext</code>
selinux	Manages current SELinux state
seboolean	Manages SELinux Booleans

Managing SELinux File Context

The essential thing to understand when working with SELinux to secure files is that the context type that is set on the file defines which processes can work with the files. The file context type can be set on a file directly, or it can be set on the SELinux policy.

When you're working with SELinux, all of its properties should be set in the SELinux policy. To do this, you use the Ansible `sefcontext` module. Setting a context type in the policy doesn't automatically apply it to files though. You still need to run the Linux `restorecon` command to do this. Ansible does not offer a module to run this command; it needs to be invoked using the `command` module.

As an alternative, you can use the `file` module to set SELinux context. The disadvantage of this approach is that the context is set directly on the file, not in the SELinux policy. As a result, if at any time default context is applied from the policy to the file system, all context that has been set with the Ansible `file` module risks being overwritten. For that reason, the recommended way to manage SELinux context in Ansible is to use the `sefcontext` module.

To be able to work with the Ansible `sefcontext` module and the Linux `restorecon` command, you also need to make sure that the appropriate software is installed on Linux. This software comes from the `policycoreutils-python-utils` RPM package, which is not installed by default in all installation patterns.

Listing 8-11 shows a sample playbook that uses this module to manage SELinux context type.

Listing 8-11 Managing SELinux Context with `sefcontext`

```
---
- name: show selinux
  hosts: all
  tasks:
  - name: install required packages
    yum:
      name: policycoreutils-python-utils
      state: present
  - name: create testfile
    file:
      name: /tmp/selinux
      state: touch
  - name: set selinux context
    sefcontext:
```



```

    target: /tmp/selinux
    setype: httpd_sys_content_t
    state: present
  notify:
    - run restorecon
  handlers:
  - name: run restorecon
    command: restorecon -v /tmp/selinux

```

In the sample playbook in Listing 8-11, the required software package is installed first. Next, a test file is created using the file module; then in the next task the **sefcontext** command is used to write the new context to the policy. If executed successfully, this task will trigger a handler to run the Linux **restorecon** command by using the command module.

Don't forget: A handler will run only if the task that triggers it generates a changed status. If the current state already matches the desired state, no changes are applied and the handler won't run!

EXAM TIP The exam assignment might not be as specific as to ask you to change a context using SELinux. You might just have to configure a service with a nondefault documentroot, which means that SELinux will deny access to the service. So also on the EX294 exam, with all tasks, you should ask yourself if this task requires any changes at an SELinux level.

Applying Generic SELinux Management Tasks

Some additional SELinux related modules are available as well. The `selinux` module enables you to set the current state of SELinux to either permissive, enforcing, or disabled. The `seboolean` module enables you to easily enable or disable functionality in SELinux using Booleans. Listing 8-12 shows an example of a playbook that uses both of these modules.

Listing 8-12 Changing SELinux State and Booleans

```

---
- name: enabling SELinux and a boolean
  hosts: ansible1
  vars:
    myboolean: httpd_read_user_content
  tasks:
  - name: enabling SELinux
    selinux:

```

```

    policy: targeted
    state: enforcing
- name: checking current {{ myboolean }} Boolean status
  shell: getsebool -a | grep {{ myboolean }}
  register: bool_stat
- name: showing boolean status
  debug:
    msg: the current {{ myboolean }} status is {{ bool_stat.stdout }}
- name: enabling boolean
  seboolean:
    name: "{{ myboolean }}"
    state: yes
    persistent: yes

```

In the sample playbook in Listing 8-12, to start with, the `selinux` module is used to ensure that SELinux is in the enforcing state. When using this module, you also have to specify the name of the policy, which in most cases is the targeted policy.

Next, the `seboolean` module is used to enable a Boolean. As you can see, this Boolean is defined as the variable `myboolean`. Before the Boolean is enabled, the `shell` and `debug` modules are used to show its current status. In Exercise 8-2 you practice working with SELinux.

Exercise 8-2 Changing SELinux Context

In this exercise you configure a more complicated playbook, running different tasks. To guide you through this process, which will prepare you for the exam in a somewhat better way, I show you a different approach this time. To start with, this is the assignment you're going to work on.

Install, start, and configure a web server that has the `DocumentRoot` set to the `/web` directory. In this directory, create a file named `index.html` that shows the message "welcome to the Exercise 8-2 webserver." Ensure that SELinux is enabled and allows access to the web server document root. Also ensure that SELinux allows users to publish web pages from their home directory.

1. Because this is a complex task, you should start this time by creating a playbook outline. A good approach for doing this is to create the playbook play header

and list all tasks that need to be accomplished by providing a name as well as the name of the task that you want to run. Create this structure as follows:

```
---
- name: Managing web server SELinux properties
  hosts: ansible1
  tasks:
    - name: ensure SELinux is enabled and enforcing
    - name: install the webserver
    - name: start and enable the webserver
    - name: open the firewall service
    - name: create the /web directory
    - name: create the index.html file in /web
    - name: use lineinfile to change webserver configuration
    - name: use sefcontext to set context on new documentroot
    - name: run the restorecon command
    - name: allow the web server to run user content
```

2. Now that the base structure has been defined, you can define the rest of the task properties. To start with, enable SELinux and set to the enforcing state:

```
---
- name: Managing web server SELinux properties
  hosts: ansible1
  tasks:
    - name: ensure SELinux is enabled and enforcing
      selinux:
        policy: targeted
        state: enforcing
```

3. You can install the web server, start and enable it, create the /web directory, and create the index.html file in the /web directory. You should be familiar with these tasks, so you can do them all in one run:

```
- name: install the webserver
  yum:
    name: httpd
    state: latest
- name: start and enable the webserver
  service:
    name: httpd
    state: started
    enabled: yes
- name: open the firewall service
  firewallld:
```

```

    service: http
    state: enabled
    immediate: yes

- name: create the /web directory
  file:
    name: /web
    state: directory
- name: create the index.html file in /web
  copy:
    content: 'welcome to the exercise82 web server'
    dest: /web/index.html
- name: use lineinfile to change webserver configuration
- name: use sefcontext to set context on new documentroot
- name: run the restorecon command
- name: allow the web server to run user content

```

4. You must use the lineinfile module to change the httpd.conf contents. Two different lines need to be changed, which you accomplish by making the following modifications:

```

- name: use lineinfile to change webserver configuration
  lineinfile:
    path: /etc/httpd/conf/httpd.conf
    regexp: '^DocumentRoot "/var/www/html"'
    line: DocumentRoot "/web"
- name: use lineinfile to change webserver security
  lineinfile:
    path: /etc/httpd/conf/httpd.conf
    regexp: '^<Directory "/var/www">'
    line: '<Directory "/web">'
- name: use sefcontext to set context on new documentroot
- name: run the restorecon command
- name: allow the web server to run user content

```

5. In the final steps, you take care of configuring the SELinux-specific settings:

```

- name: use sefcontext to set context on new documentroot
  sefcontext:
    target: '/web(/.*)?'
    setype: httpd_sys_content_t
    state: present
- name: run the restorecon command
  command: restorecon -Rv /web

```

```
- name: allow the web server to run user content
  seboolean:
    name: httpd_read_user_content
    state: yes
    persistent: yes
```

6. At this point, the complete playbook should look as follows:

```
---
- name: Managing web server SELinux properties
  hosts: ansible1
  tasks:
    - name: ensure SELinux is enabled and enforcing
      selinux:
        policy: targeted
        state: enforcing
    - name: install the webserver
      yum:
        name: httpd
        state: latest
    - name: start and enable the webserver
      service:
        name: httpd
        state: started
        enabled: yes
    - name: open the firewall service
      firewallld:
        service: http
        state: enabled
        immediate: yes
    - name: create the /web directory
      file:
        name: /web
        state: directory
    - name: create the index.html file in /web
      copy:
        content: 'welcome to the exercise82 web server'
        dest: /web/index.html
    - name: use lineinfile to change webserver configuration
      lineinfile:
        path: /etc/httpd/conf/httpd.conf
        regexp: '^DocumentRoot "/var/www/html"'
        line: DocumentRoot "/web"
```

```

- name: use lineinfile to change webserver security
  lineinfile:
    path: /etc/httpd/conf/httpd.conf
    regexp: '^<Directory "/var/www">'
    line: '<Directory "/web">'
- name: use sefcontext to set context on new documentroot
  sefcontext:
    target: '/web(/.*)?'
    setype: httpd_sys_content_t
    state: present
- name: run the restorecon command
  command: restorecon -Rv /web
- name: allow the web server to run user content
  seboolean:
    name: httpd_read_user_content
    state: yes
    persistent: yes

```

7. Run the playbook by using **ansible-playbook exercise82.yaml** and verify its output.
8. Verify that the web service is accessible by using **curl http://ansible1**. In this case, it should not show the expected welcome text. Try to analyze why. You can find the answer at the end of this chapter before the end-of-chapter. So what should you learn from this? A playbook may run without any errors, but that doesn't mean that it has produced the desired results. You should always verify!

Using Jinja2 Templates

A template is a configuration file that contains variables and, based on the variables, is generated on the managed hosts according to host-specific requirements. Using templates allows for a structural way to generate configuration files, which is much more powerful than changing specific lines from specific files. Ansible uses Jinja2 to generate templates.

Working with Simple Templates

Jinja2 is a generic templating language for Python developers. It is used in Ansible templates, but Jinja2-based approaches are also found in other parts of Ansible. For instance, the way variables are referred to is based on Jinja2.

In a Jinja2 template, three elements can be used. Table 8-4 provides an overview.

**Table 8-4** Jinja2 Template Elements

Element	Example
data	sample text
comment	{# sample text #}
variable	{{ ansible_facts['default_ipv4 ']['address '] }}
expression	{% for myhost in groups['web '] %} {{ myhost }} {% endfor %}

To work with a template, you must create a template file, written in Jinja2. Next, this template file must be included in an Ansible playbook that uses the template module. Listing 8-13 shows what a template file might look like, and Listing 8-14 shows an example of a playbook that calls the template.

Listing 8-13 Sample Template

```
# {{ ansible_managed }}

<VirtualHost *:80>
    ServerAdmin webmaster@{{ ansible_facts['fqdn'] }}
    ServerName {{ ansible_facts['fqdn'] }}
    ErrorLog logs/{{ ansible_facts['hostname'] }}-error.log
    CustomLog      logs/{{ ansible_facts['hostname'] }}-common.
log common
    DocumentRoot /var/www/vhosts/{{ ansible_facts['hostname'] }}/

    <Directory /var/www/vhosts/{{ ansible_facts['hostname'] }}>
        Options +Indexes +FollowSymlinks +Includes
        Order allow,deny
        Allow from all
    </Directory>
</VirtualHost>
```

The sample template in Listing 8-13 starts with **# {{ ansible_managed }}**. This string is commonly used to identify that a file is managed by Ansible so that administrators are not going to change file contents by accident. While processing the template, this string is replaced with the value of the **ansible_managed** variable.

This variable can be set in `ansible.cfg`. For instance, you can use `ansible_managed = This file is managed by Ansible` to substitute the variable with its value while generating the template.

As for the remainder, the template file is just a text file that uses variables to substitute specific variables to their values. In this case that is just the `ansible_fqdn` and `ansible_hostname` variables that are set as Ansible facts. To generate the template, you need a playbook that uses the template module to call the template. Listing 8-14 shows an example.

Listing 8-14 Sample Playbook

```
---
- name: installing a template file
  hosts: ansible1
  tasks:
  - name: install http
    yum:
      name: httpd
      state: latest
  - name: start and enable httpd
    service:
      name: httpd
      state: started
      enabled: true
  - name: install vhost config file
    template:
      src: listing813.j2
      dest: /etc/httpd/conf.d/vhost.conf
      owner: root
      group: root
      mode: 0644
  - name: restart httpd
    service:
      name: httpd
      state: restarted
```

In the sample playbook in Listing 8-14, the template module is used to work on the source file specified as `src`, to generate the destination file, specified as `dest`. The result is that on the managed host the template is generated, with all the variables substituted to their values.

Applying Control Structures in Jinja2 Using for

In templates, control structures can be used to dynamically generate contents. A **for** statement can be used to iterate over all elements that exist as the value of a variable. Let's look at some examples.

To start with, Listing 8-15 shows a template where a **for** statement is shown.

Listing 8-15 Exploring Jinja2 **for** Statements

```
{% for node in groups['all'] %}
host_port={{ node }}:8080
{% endfor %}
```

In this Jinja2 file, a variable with the name **host_ports** is defined on the second line (which is the line that will be written to the target file). To produce its value, the host group **all** is processed in the **for** statement on the first line. While processing the host group, a temporary variable with the name **node** is defined. This value of the **node** variable is replaced with the name of the host while it is processed, and after the host name, the string **:8080** is copied, which will result in a separate line for each host that was found. As the last element, **{% endfor %}** is used to close the **for** loop. In Listing 8-16 you can see an example of a playbook that runs this template.

Listing 8-16 Generating a Template with a Conditional Statement

```
---
- name: generate host list
  hosts: ansible2
  tasks:
  - name: template loop
    template:
      src: listing815.j2
      dest: /tmp/hostports.txt
```

As you can see, the sample playbook in Listing 8-16 uses the template as the source file and, based on the template, produces the file `/tmp/hostports.txt` on the managed host. To verify, you can use the ad hoc command **ansible ansible2 -a "cat /tmp/hostports.txt"**.

Using Conditional Statements with if

The **for** statement can be used in templates to iterate over a series of values. The **if** statement can be used to include text only if a variable contains a specific value or evaluates to a Boolean true. Listing 8-17 shows a sample template file that reacts on a variable that is set in the sample playbook in Listing 8-18.

Listing 8-17 Template Example with if

```
{% if apache_package == 'apache2' %}
  Welcome to Apache2
{% else %}
  Welcome to httpd
{% endif %}
```

Listing 8-18 Using the Template with if

```
---
- name: work with template file
  vars:
    apache_package: 'httpd'
  hosts: ansible2
  tasks:
  - template:
    src: listing817.j2
    dest: /tmp/httpd.conf
```

Using Filters

In Jinja2 templates, you can use filters. Filters are a way to perform an operation on the value of a template expression, such as a variable. The filter is included in the variable definition itself, and the result of the variable and its filter is used in the file that is generated. Table 8-5 gives an overview of some common filters. In Exercise 8-3 you practice your skills and work with templates that use a conditional statement.

Table 8-5 Common Filters Overview

Filter Example	Use
{{ myvar to_json}}	Writes the contents of myvar in JSON format
{{ myvar to_yaml}}	Writes the contents of myvar in YAML format
{{ myvar ipaddr}}	Tests whether myvar contains an IP address

EXAM TIP The Ansible documentation at <https://docs.ansible.com> contains a section with the title “Frequently Asked Questions.” In this section you can find the question “How do I loop over a list of hosts in a group, inside a template?” Read it now, and study it. The response here provides a very nice example of using conditional statements in templates, and that information might be useful on the exam.

Exercise 8-3 Working with Conditional Statements in Templates

1. Use your editor to create the file `exercise83.j2`. Include the following line to open the Jinja2 conditional statement:


```
{% for host in groups['all'] %}
```
2. This statement defines a variable with the name **host**. This variable iterates over the magic variable **groups**, which holds all Ansible host groups as defined in inventory. Of these groups, the **all** group (which holds all inventory host names) is processed.
3. Add the following line (write it as one line; it will wrap over two lines, but do not press Enter to insert a newline character):

```
{{ hostvars[host]['ansible_default_ipv4']['address'] }} {{
hostvars[host]['ansible_fqdn'] }} {{ hostvars[host]['ansible_
hostname'] }}
```

This line writes a single line for each inventory host, containing three items. To do so, you use the magic variable **hostvars**, which can be used to identify Ansible facts that were discovered on the inventory host. The **[host]** part is replaced with the name of the current host, and after that, the specific facts are referred to. As a result, for each host a line is produced that holds the IP address, the FQDN, and next the host name.

4. Add the following line to close the **for** loop:


```
{% endfor %}
```
5. Verify that the complete file contents look like the following and write and quit the file:

```
{% for host in groups['all'] %}
{{ hostvars[host]['ansible_default_ipv4']['address'] }} {{
hostvars[host]['ansible_fqdn'] }} {{
hostvars[host]['ansible_hostname'] }}
{% endfor %}
```

6. Use your editor to create the file `exercise83.yaml`. It should contain the following lines:

```
---
- name: generate /etc/hosts file
```

```

hosts: all
tasks:
- name:
  template:
    src: exercise83.j2
    dest: /tmp/hosts

```

7. Run the playbook by using **ansible-playbook exercise83.yaml**.
8. Verify the /tmp/hosts file was generated by using **ansible all -a "cat /tmp/hosts"**.

Summary

In this chapter you learned how to manipulate text files with Ansible. In the first section you learned about the most important Ansible modules that can be used. Next, you learned how to manage SELinux with Ansible. In the last part of this chapter, you read about generating configuration files using Jinja2 templates.

Exam Preparation Tasks

As mentioned in the section “How to Use This Book” in the Introduction, you have a couple of choices for exam preparation: the exercises here, Chapter 16, “Final Preparation,” and the exam simulation questions on the companion website.

Review All Key Topics

Review the most important topics in this chapter, noted with the Key Topics icon in the outer margin of the page. Table 8-6 lists a reference of these key topics and the page numbers on which each is found.



Table 8-6 Key Topics for Chapter 8

Key Topic Element	Description	Page Number
List	File module tasks	182
Table 8-4	Jinja2 Template Elements	195

Memory Tables

Print a copy of Appendix D, “Memory Tables” (found on the companion website), or at least the section for this chapter, and complete the tables and lists from memory. Appendix E, “Memory Tables Answer Key,” also on the companion website, includes completed tables and lists to check your work.

Define Key Terms

Define the following key terms from this chapter, and check your answers in the glossary:

Jinja2, template

Review Questions

1. Which module should you use to work with file system ACLs?
2. Which modules can you use to replace strings in text files based on regex? (List two.)
3. Which module should you use to retrieve file status?
4. List three tasks that are commonly executed using the file module.
5. Which module should you use to synchronize the contents of a file with the contents of a file on the control host?
6. What is wrong with using the file module to manipulate SELinux file context?
7. Which module can you use to change SELinux Booleans?
8. A playbook runs successfully, but the handler in that playbook is not triggered. What is the most common explanation?
9. How do you include a comment line in a Jinja2 template?
10. What is the `if` statement used for in Ansible templates?

Exercise Answers

After you perform all the steps in Exercise 8-2, the web server still doesn't work. Further analysis shows that the changes in `httpd.conf` have been made successfully and also that the SELinux context is set correctly. However, after you apply the changes with `lineinfile`, the web server needs to be started. You can do this either by using a handler or by moving the service task to be performed after the `lineinfile` task.

End-of-Chapter Labs

Lab 8-1: Generate an /etc/hosts File

Write a playbook that generates an `/etc/hosts` file on all managed hosts. Apply the following requirements:

- All hosts that are defined in inventory should be added to the `/etc/hosts` file.

Lab 8-2: Manage a vsftpd Service

Write a playbook that uses at least two plays to install a `vsftpd` service, configure the `vsftpd` service using templates, and configure permissions as well as SELinux. Apply the following requirements:

- Install, start, and enable the `vsftpd` service. Also open a port in the firewall to make it accessible.
- Use the `/etc/vsftpd/vsftpd.conf` file to generate a template. In this template, you should use the following variables to configure specific settings. Replace these settings with the variables and leave all else unmodified:
 - `anonymous_enable`: yes
 - `local_enable`: yes
 - `write_enable`: yes
 - `anon_upload_enable`: yes
- Set permissions on the `/var/ftp/pub` directory to mode `0777`.
- Configure the `ftpd_anon_write` Boolean to allow anonymous user writes.
- Set the `public_content_rw_t` SELinux context type to the `/var/ftp/pub` directory.
- If any additional tasks are required to get this done, take care of them.

Index

A

- ad hoc commands, 50–51, 72
 - arguments, 50
 - components, 51
 - modules, 57–58
 - output, 51–52
 - running, 53
 - running from shell scripts, 64–65
- Ansible, 6, 8. *See also* conditional statements; playbooks; SELinux; variables
- ad hoc commands, 50–51, 72
 - components, 51
 - output, 51–52
 - running, 53
 - running from shell scripts, 64–65
- architecture, 9–10
- controller node, 9, 18
 - installation, 20
- declarative approach, 11
- docs.ansible.com website, 62–63
- documentation, 63–64
- dynamic inventory scripts, 40–41
- environments, 18
- facts, 101–102
 - commonly used, 104
 - gathering, 102–104, 108
 - injected variables, 104–108
- Galaxy
 - collections, 217
 - role installation, 218–219
 - role management, 218–219
 - roles, 214, 217–218
 - website, 215–217
- handlers, 149–156
- installation methods, 20
- installing
 - on CentOS 8, 21
 - on RHEL 8, 21
 - using python-pip, 21–22
- inventory, 35
 - cache management, 235
 - commands, 37–38
 - dynamic, 39, 233–235
 - host groups, 36–38, 232–233
 - host name patterns, 232–233
 - listing hosts, 35
 - specifying host variables, 38–39
 - working with multiple inventory files, 236–237
- managed hosts, configuring, 22
- modules, 10, 50, 53–54, 308
 - assert, 268–270, 357
 - at, 338–339
 - authorized_key, 315–316
 - browsing, 58
 - command, 54
 - consulting documentation, 58
 - copy, 55
 - correcting file properties, 180
 - cron, 337
 - debug, 118, 263, 322–323
 - filesystem, 367–368
 - lvol, 366–367

- managing file attributes, 176–180
- managing SSH connections, 313
- managing sudo, 309–311
- manipulating files, 176
- mount, 368
- parted, 364–365
- ping, 56–57
- raw, 55
- service, 56
- shell, 55
- stat, 265–268
- for storage management, 354
- uri module, 264–265
- user and group management, 308–309, 311–312
- using in ad hoc commands, 57–58
- yum, 56
- parallel task execution, 237–239
- playbooks, 10, 34, 72
 - arrays, 115
 - check mode, 254–255
 - defining variables, 112
 - dictionaries, 116
 - elements, 73
 - file management, 177–187
 - host and group variables, 113–115
 - importing, 240–242
 - include files, 112–113
 - increasing output verbosity, 86–90
 - indentation, 73
 - magic variables, 117
 - multiplay, 83–85, 90–92
 - performing a dry run, 81–82
 - running, 74–77
 - tags, 271–273
 - troubleshooting, 90
 - undoing modifications, 75
 - variables, 100, 111–112
 - Vault-encrypted files, 120–121
- projects, 34–35
- and Python, 9
- RHEL 8 managed node, setting up, 23
- roles, 208–209
 - calling from playbooks, 210
 - custom, 210–213
 - dependencies, 213
 - directories, 209
 - file organization best practices, 213–214
 - locations, 209–210
 - recommendations, 214
- rules, 11
- serial task execution, 237–238
- use cases
 - configuration management, 11
 - continuous delivery, 12
 - provisioning, 12
- users
 - configuring, 23
 - managing, 323–328
 - setting up SSH, 23–25
 - sudo, 25
- variables
 - facts, 100
 - loops, 135–136
- Vault, 119, 121–122
 - managing encrypted files, 120
 - managing files with sensitive variables, 121
- when statement, 139, 148–149
 - checking for variables, 142–143
 - combining multiple conditions, 144
 - combining with loop, 145–146
 - for conditional software installation, 140–141
 - conditional tests, 141–142
 - evaluating multiple conditions, 144
 - integer checks, 144
- ansible command, 38
- Ansible Engine, 10
- Ansible Tower, 10
- ansible.cfg file
 - creating, 43
 - managing settings, 42–43

ansible-doc command, 58–60
 output elements, 60–61
 showing usage information, 62
 ansible-galaxy command, 217
 ansible-galaxy info command, 217–218
 ansible-inventory command, 37, 235–236
 ansible-playbook command, 75, 101
 output, 256–257
 output verbosity options, 257–259
 ansible-vault command, 119
 options, 120
 Appstream modules, installing, 286–287
 architecture, Ansible, 9–10
 arguments
 at module, 338–339
 set_fact, 360
 arrays, 115
 assert module, 268–270, 357
 authentication, SSH (Secure Shell),
 configuring, 24–25
 authorized_key module, 315–316
 automation, 6. *See also* Ansible
 Ansible, 8
 architecture, 9–10
 controller node, 9, 18
 declarative approach, 11
 documentation, 63–64
 environments, 18
 installation methods, 20
 installing on controller node, 20
 installing on RHEL 8, 21
 listing hosts, 35
 modules, 10, 53
 playbooks, 10, 34, 72
 projects, 34–35
 and Python, 9
 rules, 11
 specifying host variables, 38–39
 use cases, 11–12
 Ansible Engine, 10
 Ansible Tower, 10
 Chef, 8

 machine-readable code, 8
 Puppet, 8
 SaltStack, 9
 shell scripts, 6

B

best practices, file organization, 213–214
 blockinfile module, 181–182
 blocks, 163–169
 using with rescue and always
 statements, 164–165
 Booleans, 190. *See also* seboolean module
 boot process, managing, 341
 boot state, 344
 rebooting managed hosts, 342–343
 services, 345–346
 systemd targets, 341
 browsing available modules, 58
 building, 7

C

calling roles from playbooks, 210
 cat/etc/passwd command, 122–123
 CentOS 8, installing Ansible, 21
 check mode, 254–255
 templates, 261–262
 Chef, 8
 chmod+x command, 39
 code, infrastructure as, 8
 coding, 7
 combining
 multiple conditions, loop and register
 statements, 147
 when and loop statements, 145–146
 command module, 54
 commands. *See also* ad hoc commands
 ansible, 38
 ansible-doc, 58–60
 output elements, 60–61
 showing usage information, 62
 ansible-galaxy, 217
 ansible-galaxy info, 217–218

- ansible-inventory, 37, 235–236
 - ansible-playbook, 75, 101
 - output, 256–257
 - output verbosity options, 257–259
 - ansible-vault, 119
 - options, 120
 - cat/etc/passwd, 122–123
 - chmod+x, 39
 - echo password, 321
 - mkdir base, 38
 - optimizing output error formatting, 259
 - register parameter, 122–124
 - restorecon, 188
 - sefcontext, 188–189
 - ssh, 23
 - useradd, 309
 - visudo, 25
 - commonly used facts, 104
 - conditional statements
 - blocks, 163–164, 165–169
 - using with rescue and always statements, 164–165
 - changed_when, 162–163
 - combining multiple conditions, 144
 - evaluating multiple conditions, 144
 - failed_when, 158–160
 - generating Jinja2 templates, 197
 - if, 198
 - in templates, 199–200
 - using storage-related facts, 357–360
 - conditional tests, 141–142
 - file state, 178
 - configuration management, 11
 - configuring, 7
 - Ansible
 - managed hosts, 22
 - sudo, 25
 - user, 23
 - managed nodes, 297–301
 - SSH (Secure Shell), key-based user authentication, 24–25
 - swap space, 368–369
 - connectivity, troubleshooting, 275–277
 - consulting documentation, Ansible, 58
 - controller node, Ansible, 9, 18
 - copy module, 55, 184
 - copying, user SSH public key, 317
 - creating
 - ansible.cfg file, 43
 - custom facts, 110–111
 - custom roles, 210–213
 - LVM logical volume, 367
 - partitions, 364–366
 - users with encrypted passwords, 322
 - volume groups, 366–367
 - XFS file system, 368
 - cron jobs
 - managing, 337, 339–340
 - removing, 338
 - running, 337–338
 - cron module, 337
 - custom facts, 109–110
 - creating, 110–111
 - custom roles, 210
 - creating, 210–213
 - CVS (Concurrent Version System), 8
- ## D
- debug module, 118, 263, 322–323
 - defining, variables, 112
 - device names
 - managing, 360–361
 - storing in variables, 360–361
 - DevOps way of working, 7
 - dictionaries, 115
 - addressing specific keys, 116
 - discovering storage-related facts, 354
 - using ansible -m setup, 355–357
 - disk device
 - configuring swap space, 368–369
 - creating LVM logical volume, 367
 - creating LVM volume groups, 366–367
 - file system, mounting, 368

- partitions, creating, 364–366
- setting up storage, 369–373
- storing name in variables, 360–361
- docs.ansible.com website, 62–63
- documentation
 - Ansible, 63–64
 - ansible-doc command, 58–61
 - consulting, 58
- dynamic inventory, Ansible, 39
- dynamic inventory scripts, 40, 41, 233–235

E

- echo password command, 321
- encrypted files, 121
 - managing with Ansible Vault, 120
- passwords, 121–122
 - creating, 322
 - generating, 319–321
 - managing, 319

F

- facts, 100–102
 - caching, 108
 - commonly used, 104
 - custom, 109–110
 - creating, 110–111
 - gathering, 102–104, 108
 - injected variables, 104–108
 - storage-related, 354–355
 - discovering using `ansible -m setup`, 355–357
 - using in conditional statements, 357–360
- failed_when conditional, 158–160
- fetch module, 184
- file module, correcting file properties, 180
- file systems, 367
 - creating, 368
 - mounting, 368

files

- best practices, 213–214
- content management, 180–182
- creating and removing, 182–183
- including and importing, 239
- inventory, 236–237
- logging to, 259
- managing, 177–187
 - managing attributes, 176–180
 - manipulating, 176
 - moving, 184
 - properties, 180
- filesystem module, 367–368
- filters, Jinja2, 198–199
- find module, managing file contents, 180–182
- FTP-based repositories, setting up, 290–291

G

- gathering, facts, 102–104, 108
- generating, encrypted passwords, 319–321
- GitHub, 41
- GPG keys, managing, 291–292
- group variables, 113–115
- groups, managing, 308–309, 311–312

H

- handlers, 149–156, 189. *See also* tasks
 - forcing to run, 158
- host groups, Ansible, 36–38
 - wildcards, 232–233
- host variables, 113–115

I

- idempotent, 10
- if statements, 198
- importing
 - files, 239
 - playbooks, 240–242
 - task files, 242–247

include files, 112–113, 242–247
 infrastructure, as code, 8
 installing
 Ansible
 on CentOS 8, 21
 on controller node, 20
 on RHEL 8, 21
 using python-pip, 21–22
 Ansible Galaxy roles, 218–219
 Appstream modules, 286–287
 package groups, 286
 RHEL system roles, 220
 integer checks, 144
 inventory, Ansible, 35
 cache management, 235
 commands, 37–38
 dynamic, 39
 dynamic inventory scripts, 40–41
 host groups, 36–38
 host name patterns, 232–233
 listing hosts, 35
 scripts, 233–235
 specifying host variables, 38–39
 working with multiple inventory files,
 236–237
 inventory scripts, 40–41
 IT, 7
 DevOps way of working, 7

J

Jinja2, 194
 if statements, 198
 for statements, 197
 templates, 189–194
 conditional statements, 199–200
 filters, 198–199
 generating with a conditional
 statement, 197

K

key-based user authentication, setting up
 on SSH, 24–25

keys, SSH (Secure Shell), 313
 copying, 317
 creating the user, 317
 lookup plug-in, 314
 setting up, 314–316
 user management, 317–318
 key-value pairs, 78

L

lineinfile module, 181
 Linux command-line utilities, managing
 subscriptions, 293
 listing
 Ansible hosts, 35
 tags and tasks, 273–274
 lists. *See also* arrays; loops, YAML, 79
 logging, to files, 259
 loops, 134–135, 138–139
 with `_items`, 137–138
 combining with register statement, 147
 combining with when statement,
 145–146
 for multivalued variables, 136–137
 for variables, 135–136
 LVM volume groups, creating, 366–367
 lvol module, 366–367

M-N

machine-readable code, 8
 magic variables, 117
 managed hosts
 configuring, 22
 rebooting, 342–343
 managed nodes, configuring, 297–301
 managing
 Ansible Galaxy roles, 218–219
 boot process, 341
 boot state, 344
 rebooting managed hosts, 342–343
 services, 345–346
 systemd targets, 341
 cron jobs, 337, 339–340

- device names, 360–361
- files, 177–187
- GPG keys, 291–292
- SELinux properties, 187
 - file context, 188–194
- services, 336
 - systemd, 336–337
- software packages, 284, 289
 - configuring repository access, 284–285
 - package_facts module, 287
 - yum module, 285–287
- SSH connections, 313
- storage, 354
- subscriptions, 292–294
- sudo, 309–311
 - template file, 310–311
 - variables, 310
- users and groups, 308–309, 311–312, 317–318, 323–328
- mkdir base command, 38
- at module
 - arguments, 338–339
 - running commands in the future, 339
- modules, 308
 - Ansible, 10, 53, 54
 - Appstream, installing, 286–287
 - assert, 268–270, 357
 - at
 - arguments, 338–339
 - running commands in the future, 339
 - authorized_key, 315–316
 - blockinfile, 181–182
 - browsing, 58
 - command, 54
 - consulting documentation, 58
 - copy, 55, 184
 - cron, 337
 - debug, 118, 263, 322–323
 - fetch module, 184
 - file
 - correcting file properties, 180
 - creating and removing files, 182–183
 - filesystem, 367–368
 - find, managing file contents, 180–182
 - lineinfile, 181
 - lvol, 366–367
 - for managing SELinux changes, 187
 - managing software packages, 284–285
 - managing SSH connections, 313
 - managing sudo, 309–311
 - manipulating files, 176
 - mount, 368
 - package_facts, 287
 - parted, 364–365
 - creating partitions with, 365–366
 - ping, 56, 57
 - raw, 55
 - seboolean, 189–190
 - sefcontext, 188–189
 - service, 56
 - for service management, 336
 - shell, 55
 - stat, 265–268
 - managing file attributes, 176–180
 - for storage management, 354
 - synchronize, 184
 - for troubleshooting, 271
 - uri, 264–265
 - user and group management, 308–309, 311–312
 - using in ad hoc commands, 57–58
 - yum, 56
 - managing software packages, 285–287
- monitoring, 7
- mount module, 368
- mounting, file systems, 368
- moving, files, 184
- multiplay playbooks, 83–85, 90–92
- multivalued variables
 - arrays, 115

- dictionaries, 116
 - addressing specific keys, 116
 - loops, 136–139
 - with_items, 137–138
 - square brackets notation, 117
- O**
- optimizing, command output error
 - formatting, 259
 - output
 - ansible-playbook command, 256–257
 - optimizing error formatting, 259
- P**
- package groups, installing, 286
 - package_facts module, 287
 - packaging, 7
 - parallel task execution, 237–239
 - parted module, 364–365
 - partitions
 - creating, 364–365
 - creating with parted module, 365–366
 - passphrases, 24
 - passwords, encrypted, 121–122
 - creating users with, 322
 - generating, 319–321
 - managing, 319
 - ping module, 56–57
 - playbooks. *See also* roles
 - Ansible, 10, 72
 - automating RHEL host setup, 297–301
 - check mode, 254–255
 - templates, 261–262
 - conditional statements
 - blocks, 163–169
 - changed_when, 162–163
 - failed_when, 158–160
 - elements, 73
 - file management, 177–187
 - importing, 240–242
 - including and importing files, 239
 - increasing output verbosity, 86–90
 - indentation, 73
 - injected variables, 104–108
 - multiplay, 83–85, 90–92
 - performing a dry run, 81–82
 - register parameter, 122–123
 - running, 74–77
 - SELinux System Role, 222–223
 - for software management, 297
 - stat module, managing file attributes, 176–180
 - for storage management, 354
 - tags, 271–273
 - for debugging, 274–275
 - special, 274
 - tasks, 149
 - errors, 156–157
 - execution, 156
 - handlers, 149–156, 189
 - parallel execution, 237–239
 - running one by one, 260–261
 - serial execution, 237–238
 - specifying failure conditions, 158–160
 - troubleshooting, 90
 - undoing modifications, 75
 - variables, 100, 111–112
 - arrays, 115
 - defining, 112
 - dictionaries, 116
 - facts, 101–104, 108–110
 - host and group, 113–115
 - include files, 112–113
 - loops, 135–139
 - magic, 117
 - precedence, 119
 - showing, 118
 - Vault-encrypted files, 120–121
 - when statement, 139, 148–149
 - checking for variables, 142–143
 - combining multiple conditions, 144
 - for conditional software installation, 140–141

- conditional tests, 141–142
- evaluating multiple conditions, 144
- integer checks, 144
- referring to variables in, 142
- plays, 73
- practicing, YAML, 82–83
- precedence, variable, 119
- privilege escalation, setting up, 25–26
- projects, Ansible, 34–35
- provisioning, 12
- Puppet, 8
- Python, 9, 20
 - dictionaries, 115
 - dynamic inventory scripts, 40–41
 - scripts, 20
 - versions, 20
- python-pip, installing Ansible, 21–22

R

- raw module, 55
- rebooting managed hosts, 342–343
- register parameter, combining with loops, 147
- registering for the exam, 384
- releasing, 7
- removing cron jobs, 338
- repositories
 - GPG keys, 291–292
 - RPM keys, fetching, 292
 - setting up, 290–291, 294–297
 - subscriptions, managing, 292–294
- restorecon command, 188
- RHCE exam
 - exam day, 384–386
 - nondisclosure agreement, 386
 - registration, 384
 - verifying your readiness, 383
- RHEL 8
 - installing Ansible, 21
 - managed node, setting up, 23
 - managed nodes, configuring, 297–301
 - subscriptions, managing, 292–294

- system roles, 219–220
 - installing, 220
 - SELinux, 221–224
 - TimeSync, 224–225
- roles, 208–209
 - Ansible Galaxy, 214
 - installing, 218–219
 - managing, 218–219
 - requesting additional information, 217–218
- calling from playbooks, 210
- custom, 210–213
- dependencies, 213
- directories, 209
- file organization best practices, 213–214
- locations, 209–210
- recommendations, 214
- RHEL, 219–220
 - installing, 220
 - SELinux System Role, 221–224
 - TimeSync System Role, 224–225
- RPM keys, fetching, 292
- rules, Ansible, 11
- running
 - ad hoc commands, 53
 - from shell scripts, 64–65
 - cron jobs, 337–338
 - playbooks, 74–77
 - tasks, 260–261

S

- SaltStack, 9
- scripts
 - inventory, 40, 41, 233–235
 - Python, 20
- seboolean module, 189–190
- security
 - Ansible Vault, 121–122
 - managing encrypted files, 120
 - privilege escalation, 25–26
 - Vault-encrypted files, 120–121

- sefcontext module, 188–189
 - SELinux
 - changing context, 191–194
 - managing properties, 187
 - file context, 188–194
 - SELinux System Role, 221
 - playbooks, 222–223
 - rebooting managed hosts, 223–224
 - variables, 222
 - serial task execution, 237–238
 - servers, 6
 - service module, 56
 - services
 - and boot process, 345–346
 - managing, 336
 - systemd, 336–337
 - set_fact argument, 360
 - setting up
 - Ansible, 23–25
 - privilege escalation, 25–26
 - repositories, 290–291, 294–297
 - RHEL 8 managed node, 23
 - SSH keys, 314–316
 - storage, 369–371, 372–373
 - test networks, 19
 - users, Ansible, 323–328
 - settings, Ansible, 42–43
 - shell module, 55
 - shell scripts, 6
 - running ad hoc commands, 64–65
 - software packages
 - managing, 284–285, 289
 - package_facts module, 287
 - yum module, 285–287
 - repositories, setting up, 290–291
 - special tags, 274
 - SSH (Secure Shell), 18
 - key-based user authentication,
 - configuring, 24–25
 - keys
 - copying, 317
 - creating the user, 317
 - lookup plug-in, 314
 - setting up, 314–316
 - user management, 317–318
 - managing, 313
 - public/private keys, 24
 - stat module, 265–268
 - managing file attributes, 176–180
 - for statements, 197
 - storage-related facts, 354–355
 - discovering, 354
 - using in conditional statements,
 - 357–360
 - storing
 - disk device name in variables, 360–361
 - roles, 209–210
 - strings, YAML, 79–80
 - subscriptions, managing, 292–294
 - sudo
 - configuring for the Ansible user, 25
 - managing, 309–311
 - template file, 310–311
 - variables, 310
 - swap space configuring, 368–369
 - synchronize module, 184
 - syntax, YAML, verifying, 80–81
 - system roles
 - SELinux, 222–224
 - TimeSync, 224–225
 - systemd services, 336–337
 - systemd targets, managing, 341
- ## T
- tags, 271–273
 - for debugging, 274–275
 - listing, 273–274
 - special, 274
 - tasks, 149
 - errors, 156–157
 - execution, 156
 - handlers, 149–156, 189
 - forcing to run, 158
 - includes and imports, 242–247

- listing, 273–274
 - parallel execution, 237–239
 - running, 260–261
 - serial execution, 237–238
 - specifying failure conditions, 158–160
 - templates
 - Jinja2, 189–194
 - conditional statements, 199–200
 - filters, 198–199
 - generating with a conditional statement, 197
 - for managing sudo, 310–311
 - using in check mode, 261–262
 - test networks, setting up, 19
 - testing, 7
 - TimeSync System Role, 224–225
 - troubleshooting
 - assert module, 268–270
 - connectivity, 275–277
 - debug module, 263
 - with modules, 263
 - playbooks, 90
 - stat module, 265–268
 - tasks, 156–157
 - uri module, 264–265
- U**
- undoing playbook modifications, 75
 - uri module, 264–265
 - use cases, Ansible, configuration
 - management, 11
 - useradd command, 309
 - users
 - creating with encrypted passwords, 322
 - managing, 308–309, 311–312, 323–328
 - managing with SSH keys, 317–318
- V**
- variables, 100, 111–112
 - checking for, 142–143
 - defining, 112
 - encrypted files, 121
 - facts, 100–102
 - caching, 108
 - commonly used, 104
 - custom, 109–110
 - gathering, 102–104, 108
 - injected variables, 104–108
 - storage-related, 354–357
 - storage-related, discovering, 354
 - host and group, 113–115
 - for importing and including files, 243
 - include files, 112–113
 - loops, 135–139
 - with_items, 137–138
 - magic, 117
 - for managing sudo, 310
 - multivalued
 - arrays, 115
 - dictionaries, 116
 - square brackets notation, 117
 - precedence, 119
 - SELinux System Role, 222
 - showing, 118
 - storing disk device name in, 360–361
 - when statements, 142
- verbosity options, ansible-playbook
- command, 257–259
- verifying, YAML syntax, 80–81
- versions, Python, 20
- visudo command, 25
- volume groups, creating, 366–367
- W**
- websites
- Ansible Galaxy, 215–217
 - docs.ansible.com, 62–63
- when statement, 139, 148–149
- blocks, 163–169
 - using with rescue and always statements, 164–165
 - checking for variables, 142–143
 - for conditional software installation, 140–141

- conditional tests, 141–142
 - evaluating multiple conditions, 144
 - integer checks, 144
 - referring to variables in, 142
- wildcards, host group, 232–233

X-Y-Z

- XFS file system, creating, 368
- YAML, 8, 10, 73, 77

- indentation, 77
- key-value pairs, 78
- lists, 79
- practicing, 82–83
- strings, 79–80
 - verifying syntax, 80–81
- yum module, 56
 - managing software packages, 285–287
- ZeroMQ message queue, 9