

Using Ansible for Automation in IBM Power Environments

Tim Simon

Jose Martin Abeleira

Shahid Ali

Vijaybabu Anaimuthu

Sambasiva Andaluri

Marcelo Avalos Del Carpio

Thomas Baumann

Ivaylo Bozhinov

Carlo Castillo

Rafael Cezario

Stuart Cunliffe

Nilabja Haldar

Munshi Hafizul Haque

Subha Hari

Andrey Klyachkin

Osman Omer

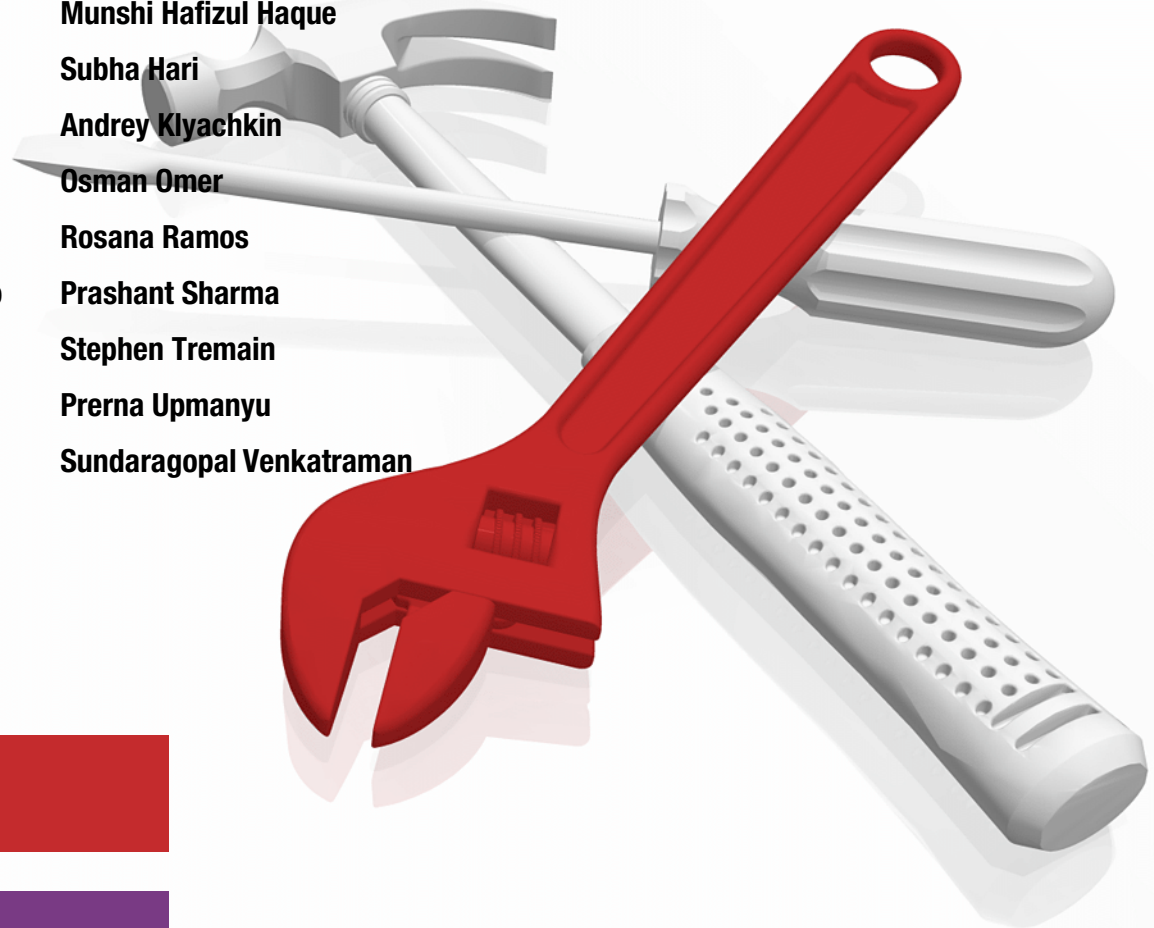
Rosana Ramos

Prashant Sharma

Stephen Tremain

Prerna Upmanyu

Sundaragopal Venkatraman



 **Cloud**

Power Systems



IBM Redbooks

Using Ansible for Automation in IBM Power Environments

November 2023

Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

First Edition (November 2023)

This edition applies to the following product versions:

Ansible Core 2.14

Ansible 2.10.8

Ansible extension v2.6.92 and v2.7.98

Ansible Automation Platform (AAP) v2.4-1.2 (ppc64le)

AIX 7.2 TL5, AIX 7.3 TL1

IBM i 7.3 TR13, IBM i 7.4 TR7, IBM i 7.5

IBM i Modernization Engine for Lifecycle Integration (Merlin) 1.0

Red Hat Enterprise Linux Server 7.9 (ppc64 - Big Endian)

Red Hat Enterprise Linux Server 8.4 (ppc64le)

Red Hat Enterprise Linux Server 9.2 (ppc64le)

SLES 15 SP 5 (ppc64le)

Ubuntu 20.04.6 (ppc64le)

Ubuntu 22.04.3 (ppc64le)

VisualStudio Code v1.83.0

PowerVM Virtual I/O Server 3.1.3 and Virtual I/O Server 3.1.4

PowerVM Virtual I/O Server 4.1.0.10

Hardware Management Console V8R8.7, V9R1 or later

VisualStudio Code v1.83.0

© Copyright International Business Machines Corporation 2023. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

This document was created or updated on August 29, 2024.

Contents

Notices	ix
Trademarks	x
Preface	xi
Authors	xi
Now you can become a published author, too!	xv
Comments welcome	xv
Stay connected to IBM Redbooks	xvi
Chapter 1. Introduction to Ansible and IBM Power	1
1.1 Why automation	2
1.1.1 Orchestration versus automation	3
1.2 Automation tools and techniques	4
1.2.1 Common IT automation tools	4
1.3 Understanding Ansible: A Powerful Automation Tool	5
1.3.1 Options for implementing Ansible	6
1.3.2 Red Hat Ansible Automation Platform	10
1.3.3 Event-Driven automation	14
1.3.4 Infrastructure as code - Integration of Ansible and Terraform	16
1.3.5 Provisioning	18
1.3.6 Patch management	19
1.3.7 Security and compliance	19
1.3.8 Configuration management	21
1.3.9 Business continuity	22
1.3.10 Application development	22
1.4 Introduction to IBM Power	23
1.4.1 Power processors and architecture	25
1.4.2 PowerVM and virtualization	26
1.4.3 Supported operating systems	27
1.4.4 Key benefits of IBM Power compared to x86 servers	28
1.5 Ansible for Power	29
1.5.1 Ansible for Linux on Power	30
1.5.2 Ansible for AIX	38
1.5.3 Ansible for IBM i	42
1.5.4 Ansible for IBM Power Hardware Management Console	50
1.5.5 Ansible for Power Virtual I/O server	52
1.5.6 Ansible for IBM Power Virtual Server	53
1.5.7 Ansible for applications	55
Chapter 2. Ansible architecture and design	59
2.1 Ansible architecture and components	60
2.1.1 Controller and client functions	61
2.2 Understanding Ansible's declarative language	62
2.2.1 YAML Structure	62
2.2.2 Jinja2	64
2.3 Understanding Ansible Inventory	65
2.3.1 Overview of Ansible inventory	66
2.3.2 Overview of dynamic inventory	68
2.4 Ansible tasks, playbooks and modules	75

2.4.1	Creating Ansible playbooks	76
2.5	Ansible roles and collections	82
2.5.1	Understanding roles in Ansible	82
2.5.2	Creating and structuring Ansible roles	83
2.5.3	Sharing and reusing roles in multiple playbooks	85
2.5.4	Role dependencies and role-based variables	86
2.5.5	Using collections	87
2.6	Preferred practices for playbook and role design	88
2.6.1	Writing modular and reusable playbooks	89
2.6.2	Using Ansible Galaxy for role management	89
2.7	Versioning and documenting playbooks and roles	91
2.8	Testing and validating playbooks and roles	95
Chapter 3. Getting started with Ansible		99
3.1	Architecting your Ansible environment	100
3.1.1	Start simple – Ansible Core and Ansible Community	100
3.1.2	Scaling up – Ansible Automation Platform	101
3.1.3	Enterprise-ready environment	114
3.1.4	Develop an “automation first” attitude	116
3.2	Choosing the Ansible controller node	117
3.3	Installing your Ansible control node	117
3.3.1	Linux as an Ansible controller	118
3.3.2	AIX as an Ansible controller	130
3.3.3	IBM i as an Ansible controller	139
3.4	Preparing your systems to be Ansible clients	144
3.4.1	Linux as Ansible-managed client	144
3.4.2	AIX as Ansible-managed client	145
3.4.3	IBM i as Ansible-managed client	146
3.4.4	VIOS as Ansible-managed client	154
3.4.5	Red Hat OpenShift as Ansible-managed client	162
3.4.6	IBM Power Hardware Management Console as Ansible-managed client	173
Chapter 4. Automated Application Deployment on Power Servers		183
4.1	Deploying and managing applications using Ansible on Power servers	184
4.2	Automated Application Deployment on Power Servers	184
4.2.1	Ansible Content for IBM Power Systems	184
4.2.2	IBM AIX, IBM i, and Linux on Power Collections for Ansible	184
4.3	Deploying a simple Node.js application	185
4.4	Orchestrating multi-tier application deployments	186
4.4.1	Orchestration in the world of Kubernetes	186
4.5	Continuous integration and deployment pipelines with Ansible	186
4.5.1	CI/CD using Ansible for IBM i	186
4.6	Oracle DB automation on Power Systems	188
4.6.1	Why businesses opt for AIX to host their databases	188
4.6.2	Automating deployment of a single node Oracle database with Ansible	189
4.6.3	Automating deployment of Oracle RAC with Ansible	193
4.6.4	Automating Oracle DBA operations	204
4.7	SAP automation	212
4.7.1	Red Hat Enterprise Linux System Roles for SAP	214
4.7.2	Using the SAP LinuxLab Automation	217
Chapter 5. Infrastructure as Code Using Ansible		227
5.1	IBM Power Virtualization Center	228
5.1.1	Using the OpenStack Cloud modules	228

5.1.2 Using the URI modules to interact with PowerVC API services	241
5.2 IBM Power Virtual Server (PowerVS)	250
5.2.1 Using the IBM Cloud collection for PowerVS	251
5.2.2 Using the URI module for PowerVS	252
Chapter 6. Day 2 Management Operations	265
6.1 Introduction to Day 2 operations	266
6.2 Day 2 operations in Linux servers	268
6.2.1 Storage	269
6.2.2 Security and compliance	274
6.2.3 Fixes and Upgrades	279
6.2.4 Configuration tuning	282
6.3 Day 2 operations in AIX environments	285
6.3.1 Storage	285
6.3.2 Security	296
6.3.3 Fixes	301
6.3.4 Configuration tuning	305
6.4 Day 2 operations in IBM i environments	307
6.4.1 Storage	307
6.4.2 Security and compliance	309
6.4.3 Fix management	311
6.4.4 Configuration tuning	312
Chapter 7. Future Trends and Directions	315
7.1 Ansible and IBM Power Systems Roadmap	316
7.1.1 Working closely with the IBM Power collections and their contents	316
7.2 Roadmap for Ansible automation in the Power ecosystem	317
7.2.1 Ansible Automation Platform on IBM Power	317
7.2.2 Visual Studio Code	318
7.2.3 IBM watsonx Code Assistant for Red Hat Ansible Lightspeed	325
Appendix A. Unveiling IBM i Merlin	331
A.1 Introduction	332
A.1.1 What is Merlin	332
A.1.2 The role of Merlin in the IBM i market	333
A.1.3 Merlin's problem-solving impact	334
A.1.4 Benefits of Merlin for IBM i modernization	334
A.1.5 Decades of collaboration: IBM and ARCAD	335
A.1.6 Components of Merlin	335
A.1.7 Comprehensive overview of Merlin content	337
Ansible integration for IBM i lifecycle management via Merlin	340
A.1.8 The business demands for DevOps on IBM i	343
A.1.9 Merlin for IBM i developers	356
A.1.10 Merlin requirements	366
Related publications	369
IBM Redbooks	369
Help from IBM	369

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at <https://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks or registered trademarks of International Business Machines Corporation, and might also be trademarks or registered trademarks in other countries.

AIX®	IBM Security®	Rational®
DB2®	IBM Sterling®	Redbooks®
DS8000®	IBM Z®	Redbooks (logo)  ®
IBM®	Instana®	SoftLayer®
IBM Cloud®	Passport Advantage®	Sterling™
IBM Cloud Pak®	POWER®	System z®
IBM Consulting™	PowerHA®	SystemMirror®
IBM FlashSystem®	PowerVM®	Turbonomic®
IBM Instana™	QRadar®	WebSphere®

The following terms are trademarks of other companies:

Intel, Intel Xeon, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

ITIL is a Registered Trade Mark of AXELOS Limited.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Ansible, OpenShift, Red Hat, are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Preface

This IBM Redbooks publication will help you install, tailor and configure an automation environment using Ansible in an IBM Power server environment. Ansible is a versatile and easy to use IT automation platform that makes your applications and systems easier to deploy and maintain. With Ansible you can automate almost anything: code deployment, network configuration, server infrastructure deployment, security and patch management, and cloud management. Ansible is implemented in an easy to use and human readable language (YAML) and uses SSH to connect to the managed systems, hence with no agents to install on remote systems.

Ansible is an Open Source solution that is gaining market share in the automation workspace as it can help you automate almost anything. This IBM Redbooks publication will show you how to integrate Ansible to manage all aspects of your IBM Power infrastructure, including server hardware, the hardware management console, PowerVM, PowerVC, AIX, IBM i, and Linux on Power. We provide guidance on where to run your Ansible automation controller nodes and demonstrate how it can be installed on any operating system supported on IBM Power and also show you how to set up your IBM Power infrastructure components to be managed using Ansible.

This publication is intended for use by anyone who is interested in automaton using Ansible whether they are just getting started or if they are experts on Ansible and want to understand how to integrate IBM Power into their existing environment.

Authors

This book was produced by a team of specialists from around the world in conjunction with IBM Redbooks.

Tim Simon is an IBM Redbooks® Project Leader in Tulsa, Oklahoma, USA. He has over 40 years of experience with IBM primarily in a technical sales role working with customers to help them create IBM solutions to solve their business problems. He holds a BS degree in Math from Towson University in Maryland. He has worked with many IBM products and has extensive experience creating customer solutions using IBM Power, IBM Storage, and IBM System z throughout his career. He currently lives in Tulsa, Oklahoma where he enjoys spending time with his grandchildren.

Jose Martin Abeleira is a Senior Systems and Storage Administrator at DGI (Uruguay Taxes Collection Agency). A former IBMer, he is a Gold Redbooks Author, Certified Consulting IT Specialist, and IBM Certified Systems Expert Enterprise Technical Support for IBM AIX and Linux in Montevideo, Uruguay. He worked with IBM for 8 years and has 18 years of AIX experience. He holds an Information Systems degree from Universidad Ort Uruguay. His areas of expertise include IBM Power, AIX, UNIX, and LINUX, Live Partition Mobility (LPM), IBM PowerHA® SystemMirror®, SAN and Storage on IBM DS line, V7000, HITACHI HUSVM, and G200/G400/G370/E590. He teaches Systems Administration in the Systems Engineer career at the Universidad Catolica del Uruguay, and he teaches Infrastructure Administration in the Computer Technologist career created by the joint venture between Universidad del la Republica Uruguay, Universidad del Trabajo del Uruguay and Universidad Tecnologica.

Shahid Ali is a Cloud Solution Lead for MEA Region. At the time of this publication, he is based in Riyadh, Saudi Arabia, and leading hybrid multi-cloud solutions in MEA region. He is an

experienced Enterprise Architect and joined IBM around 5 years back as an Enterprise Architect. He has 28 years of experience as an architect and consultant. Before joining IBM, he has provided consultancy services in some of the largest projects in Saudi Arabia in Ministries of Interior, Education and Labor and related organizations. These projects produced nationwide solutions for fingerprinting, country-wide secure networks, smart ID cards, e-services portals, enterprise resource planning systems, and massive open online courses platforms. He has several IBM and industry certifications and is also a member of IBM Academy of Technology.

Vijaybabu Anaimuthu is a Technical Consultant at IBM Systems Experts Labs in India. He holds a bachelor's degree (BE) in Electrical and Electronics Engineering from Anna University, Chennai. He has over 15 years of experience working with customers designing and deploying solutions on IBM Power server and AIX. He focuses on areas such as IT Infrastructure Enterprise Solutions, technical enablement and implementations relative to IBM Power servers, Enterprise Pools, performance, and automation. His areas of expertise include capacity planning, migration planning, system performance and automation.

Sambasiva Andaluri (Sam) is an experienced Developer turned Solution Architect Leader with over 30 years of experience. For the past decade, he has been a pre-sales and post-sales solution architect for trading systems at Fidessa, a pre-sales solution architect at AWS and as an SRE onboarding ISVs for Google marketplace at a partner. He brings multifaceted experience to the table, is a continuous learner, and is a strong supporter of STEM. In his free time, he coaches K-12 students for FIRST LEGO league competitions, inspiring the young minds to take up STEM careers.

Marcelo Avalos Del Carpio is a Cloud Architect at Kyndryl Consult in Uruguay with over 9 years of experience in Information Technology. A former IBM leader, he has specialized in deploying IBM technical solutions for key accounts across South America and North America. He holds an Electronic Systems Engineering degree from Escuela Militar de Ingeniería, Bolivia, and a master's in Project Management from GSPM UCI, Costa Rica. He is certified by The Open Group, and specializes in IT infrastructure, cloud platforms, and DevOps, drawing from frameworks such as PMI, ITIL, and TOGAF.

Thomas Baumann is Senior Systems Engineer and Managing Director of ACP IT Consulting GmbH (formerly: tiri GmbH) in Hamburg, Germany, which is an IBM Business Partner and a Red Hat Premier Partner. He has over 30 years of experience in computer technology, and is also a trainer for IBM Software, Ansible Automation, Linux/Cloud, and Security/Threat Management. His main focus is creating smart and cool solutions which fit the customer needs, always from the lens of restorability, workability and operability.

Ivaylo Bozhinov is a Power Systems subject matter expert (SME) at IBM Bulgaria. His main area of expertise is solving complex hardware and software issues on IBM Power Systems products, IBM AIX, VIOS, HMC, IBM i, PowerVM®, and Linux on Power Systems servers. He has been with IBM since 2015 providing reactive break-fix, proactive, preventative, and cognitive support.

Carlo Castillo is a Client Services Manager for IBM Power for Right Computer Systems, an IBM Business Partner and Red Hat partner in the Philippines. He has over 32 years of experience in pre-sales and post-sales support, designing full IBM infrastructure solutions, creating pre-sales configurations, performing IBM Power installation, implementation and integration services, and providing post-sales services and technical support for customers, as well as conducting presentations at customer engagements and corporate events. He was the very first IBM-certified AIX Technical Support engineer in the Philippines in 1999. As training coordinator during RCS' tenure as an IBM Authorized Training Provider from 2007 to 2014, he also administered the IBM Power Systems curriculum, and conducted IBM training classes covering AIX, PureSystems, PowerVM, and IBM i. He holds a degree in Computer Data Processing Management from the Polytechnic University of the Philippines.

Rafael Cezario is a Senior Solutions Engineer at Blue Trust, an IBM business partner in Brazil. Previously he was an employee of IBM where he worked as a Pre-Sales technical resource on IBM Power servers. He has 19 years of IT experience having worked on various infrastructure projects including design, implementation, demonstration, installation and integration of the solutions. He has worked with a wide variety of existing software on the IBM Power platform such as PowerVM implementations including SEA and vNIC, PowerVC, PowerSC, OpenShift, Ansible, and NIM Server to list a few. During his career at IBM, he served as a consultant for large clients with IBM Power and AIX and performed pre-sales and post-sales activities as well as doing presentations and demonstrations for clients. He has worked in several areas of infrastructure during his career, becoming certified in several of these technologies, Cisco CCNA, Nutanix NCA, IBM AIX, He holds a degree in Electrical Engineering with a specialization in Telecommunications from the Instituto de Ensino Superior de Brasília (IESB).

Stuart Cunliffe is a solution engineer within IBM Technology Expert Labs in the UK, specializing in IBM Power systems and helping customers exploit the most value out of their Power infrastructure. He has worked for IBM since graduating from Leeds Metropolitan University in 1995, and has held roles in IBM Demonstration Group, Global Technologies Services (GTS) System Outsourcing, eBusiness hosting, and ITS. A key area of his expertise is helping customers design and deliver automation across their IBM Power estate with solutions involving tools such as Red Hat Ansible, HashiCorp Terraform and IBM PowerVC.

Nilabja Haldar is an experienced Cloud Architect/SRE and certified AWS/GCP/Azure/IBM Cloud/OpenShift solution Architect, having 15 years of experience in various IT domain like public and Hybrid multi cloud, Technical Consulting, Solution designing, Architecting, Implementation, Transformation & Migration, Data Center consolidation for domestic and global organizations. He is working in IBM Consulting™ as an Infrastructure and Cloud architect, Devops and SRE. He has completed BTech in Computer Science and his technical skills cover hybrid cloud, GCP/Azure/IBM Cloud®, Kubernetes, OpenShift, Devops, security, observability, integration and open-source software.

Munshi Hafizul Haque is a Senior Platform Consultant at Red Hat in Kuala Lumpur, Malaysia. Munshi is an experienced technologist in engineering, design, and architecture of PaaS and cloud infrastructures. At the time of this publication, he is part of the Red Hat Consulting Services team where he helps organizations adopt automation, container technology and DevOps practices. Before that, he worked for IBM as a senior consultant with IBM Systems Lab Services in Petaling Jaya, Malaysia, where he took part in various projects with different people in different ASEAN countries, and as a specialist in IBM Power Systems and associated enterprise edition technology

Subha Hari is a Senior Delivery Consultant from IBM Technology Expert Labs (Sustainability Software) in Bangalore, India. She has over 19 years of experience, primarily in Performance Testing of IBM Sterling™ Order Management suite of applications, Production Performance Health Checks, Sizing & HA/DR activities. She holds a Masters degree (Master of Computer Applications) from Bharathidasan University, Trichy, India. Subha has led various initiatives on automation using Ansible, Python, Shell scripting etc. Her areas of expertise include pre-sales, performance testing/benchmarking, upgrade and modernization of IBM Sterling suite of products.

Andrey Klyachkin is a solution architect at eNFence, an IBM business partner in Germany. He has more than 25 years experience in UNIX systems, designing and supporting AIX® and Linux systems for different customers all over the world. He is co-author of many IBM AIX and IBM Power certifications, IBM Champion and IBM AIX Community Advocate. He is also a Red Hat Certified Engineer and Red Hat Certified Instructor. Andrey blogs a lot about IBM AIX automation using Ansible on LinkedIn where you can always find and ask him questions. Another way to ask him a question is to meet at international and local events about IBM Power like IBM TechXchange, GSE and Common Europe Congress.

Osman Omer is a senior IT Managing Consultant based in Qatar. He is in his 20th year of his IBM tenure, the first half of which he was based in Rochester, Minnesota where he worked as a software engineer for 8 years before joining Lab Services. All his IBM tenure has been on IBM Systems management, cloud solutions and automation services. His first project was porting IBM i to be managed by HMC, then IBM i OS enablement for system management, tooling, Systems Director, VMControl and PowerVC. As a Lab Services consultant, he enables IBM customers on the products he used to develop. Upon transitioning to Qatar, he became an integral member of the MEA (middle east and Africa) team owning cloud and automation services delivery in the region. He is currently acting as the EMEA Power Services Delivery Practice Leader in addition to his consulting and leadership responsibilities. Osman holds a master degree in Computer Science from South Dakota State University.

Rosana Ramos is a Security Architect at IBM Systems BISO Organization. She holds a bachelor's degree in Computer Engineering from Universidad de Guadalajara México and a master's in Computer Science from Universidad Autonoma de Guadalajara. She has more than 10 years of experience in Linux and Unix system administration, she has specialized on implementation of security best practices and system hardening. She is certified as CISSP, CEH, CRISC and by the Open Group as Master Certified Technical Specialist.

Prashant Sharma is an IBM Power Brand Technical Specialist based in Singapore. He holds a degree in Information Technology from University of Teesside, England. He has over 12 years of experience in IT Infrastructure Enterprise Solutioning, pre-sales, client, and partner consultation, technical enablement and implementations relative to IBM Power servers, IBM i and IBM Storage.

Stephen Tremain has been with IBM for 17 years, and currently works as a Software Engineer at IBM Security® - Australia Development Lab on the Gold Coast in Queensland, Australia. Before joining IBM, Stephen worked as a UNIX System Administrator with an investment bank for 10 years, and also worked in the education and research sectors. Stephen graduated from the University of New England in Australia, with a BS and a Graduate Diploma in Agricultural Sciences.

Perna Upmanyu is a Software Performance Analyst in the Cognitive Systems Power Servers performance team in India. She holds an M.Tech degree in Software Systems from BITS Pilani. Perna has over 15 years of experience working with customers designing and deploying solutions on IBM Power server. She focuses on areas such as Automation and Data Lakes-based design and deployments. Perna's areas of expertise include system performance, availability, and automation.

Sundaragopal Venkatraman (Sundar) is a Red Hat Industry Specialist. He has diversified skills on Hybrid Cloud Automation, Application Migration, and Modernization of Red Hat and IBM portfolios. Sundar has over 23 years of experience working closely with customers to overcome business challenges by leveraging technologies. A prolific author, he has been recognized as "Platinum Author" for IBM Redbooks publications. He holds multiple patents and an Invention Plateau holder. He has delivered key notes on WW conferences on Technology Transformation & Modernization. He is a co-chair for the IT specialist board in the Asia-Pacific region.

Thanks to the following people for their contributions to this project:

Sukanta Basak, Senior Manager - Solution Architecture Partner Ecosystem
Red Hat, Bengaluru, India

Jitendra Singh, Specialist Solution Architect Automation & SAP
Red Hat, Bengaluru, India

Anant Dhar, Senior Ecosystem Solution Architect,
Red Hat, Bengaluru, India

Dr Manoj Kumar Jain, Automation Specialist Solution Architect,
IBM, India

Kanan Ganjoo, Customer Success Manager Architect - Observability,
IBM, Bangalore, India

Bhargavaram Akula, WW ISV Engineering,
IBM, Hyderabad, India

Shiva Laveti, Systems Engineer - ISV Engineering Infrastructure,
IBM, Bangalore, India

Benoit Marolleau, Senior Solution Architect - IBM Client Engineering EMEA
IBM, Montpellier, France

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an IBM Redbooks residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:
ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbooks@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, IBM Redbooks
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ Find us on LinkedIn:
<https://www.linkedin.com/groups/2130806>
- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:
<https://www.redbooks.ibm.com/subscribe>
- ▶ Stay current on recent Redbooks publications with RSS Feeds:
<https://www.redbooks.ibm.com/rss.html>



1

Introduction to Ansible and IBM Power

This chapter starts with a discussion about the need for automation in today's complex IT environments and describes some of the technologies and tools that are available to bring the benefits of automation to your business. We describe Ansible, and why it is considered as the most versatile automation solution for your needs.

Also in this chapter, we discuss IBM Power, IBM's powerful and robust midrange server platform. We provide an overview of IBM Power as one of the leading enterprise server architectures in the market and show how your IBM Power servers can be automated using the same Ansible tools that you may already be using for other servers, storage devices, and networking components in your environment.

We also explore how Ansible automation can reshape the IT management landscape providing an end-to-end automation platform to configure systems, deploy software, and orchestrate workflows on IBM Power. We delve into provisioning, patch management, security, configuration, business continuity, disaster recovery, application development, and much more, showcasing how Ansible and IBM Power harness the combined potential of cutting-edge technology and a highly-configurable automation platform.

This chapter covers the following:

- ▶ Why automation
- ▶ Automation tools and techniques
- ▶ Understanding Ansible: A Powerful Automation Tool
- ▶ Introduction to IBM Power
- ▶ Ansible for Power

1.1 Why automation

The dictionary defines automation as “the technique of making an apparatus, a process, or a system operate automatically.” We define automation as “the creation and application of technology or programs to replace repeatable tasks with minimal human intervention.”

Automation in everyday life has been a staple for many years, for example:

- Automatic dishwashers do our dishes and automatic washers and dryers clean our clothes.
- Robotic machines do repetitive tasks in manufacturing.
- Machines automatically “call home” when an error is detected.
- Features in your automobile automatically check and report on safety issues and can even allow the car to drive itself.

Automation is designed to do simple repetitive tasks that consume time and energy in everyday life and makes our life easier. However, automation has become a necessity in today’s information technology world as the number of components that need to be managed is increasing exponentially.

Below are some benefits of Automation:

- ▶ Carry out processes that are difficult to be done manually.

IT automation tools helps us to do different kinds of tasks that would be difficult to carry out manually. For example, provisioning and deploying environments manually is a laborious process which requires highly skilled personnel with hands-on knowledge. Using automation to apply Infrastructure as a Code (IaC) enables your IT team to provide self-service capabilities to developers. It delivers preapproved resources and configurations very quickly, on demand, without manual intervention.

- ▶ Create solutions that work consistently across different technologies or cloud platforms.

IT team tasks are quite complex in hybrid and multicloud environments. Every cloud provider has their own tools and methods, which rarely works well with each other. This makes life difficult for IT teams as they have to manage each cloud separately and also in a different manner. IT automation tools comes to rescue here. Automation assets can be created by codifying resources across all clouds. It can offer a single API for a given operation, regardless of the cloud platform and this helps IT teams operate more efficiently and effectively.

- ▶ Keep pace with increasing infrastructure needs.

Infrastructure needs grow rapidly so often IT teams strive hard to manage new requirements specially if the staffing levels are low. IT automation tools will help the IT team handle this situation by eliminating or streamlining a vast array of manual tasks and processes.

- ▶ Integrate and deploy applications quickly with zero downtime.

As most of the organizations are moving towards modernization, fast and reliable application development is crucial. Continuous Integration/Continuous Deployment (CI/CD) approach helps organizations deliver applications more swiftly and with minimal errors. By adopting a CI/CD pipeline that applies automation throughout the application life cycle – including integration, testing, delivery and deployment – the teams can produce stable applications more quickly and with zero downtime.

- ▶ Produce trusted, secured and compliant applications.

When there is an automated CI/CD pipeline with security gates in place, one can be assured that a trusted software is produced for deployment. This automation helps streamline daily operations and integrate security and compliance in the processes from the beginning.

- ▶ Construct remediation processes.

When there is a security breach, they should be detected and contained as quickly as possible. If there are multiple systems and platforms, then it will be a complicated task to apply fixes manually and they can be error-prone too. Every second matters during a security breach, and automation helps security team apply remediation to affected systems across all environments more quickly, with fewer chances for errors.

- ▶ Get more time to focus on competitive and novel initiatives.

Most of the IT tasks can be automated to some extent. Automating even a few of them can help reduce the amount of time IT teams spend on manual processes. The more that automation is used, the more time they will have to work on futuristic and innovative projects.

- ▶ Eliminate human errors.

The manual routine tasks can be error-prone. With automation, these can be eliminated and one can expect predictable and consistent results.

- ▶ Get to know operational complexities and costs.

While automation helps to streamline and manage complex tasks, it can also provide operational analytics which can help to understand and reduce the costs involved.

- ▶ Transforms organization.

When automation becomes the second nature of an organization, the teams can save time, money and have more time to work on strategic initiatives. It increases productivity and decreases costs due to human errors. It increases employee satisfaction because manual and repetitive tasks are boring and laborious. Happier employees, less error-prone processes, cost and time savings makes a successful organization.

1.1.1 Orchestration versus automation

Automation generally applies to doing a single process, or task, or a small number of related tasks, For example: checking if a number of systems need updates, or checking if updates have been performed.

Orchestration refers to managing multiple automated tasks in order to create a dynamic workflow, As an example using the same case as above: check if my systems need updates, if the update is needed perform the update needed on each system, and if a reboot is needed, perform the reboot or inform the user that a reboot is needed or scheduled for later.

The two concepts are closely related and at times the line between the two concepts can be blurry. The biggest differentiator is that orchestration takes a set of automated tasks and groups them together, checking for values before and after a task completes, checking the results for each task (was it successful or not), and adding intelligence to the workflow and adapting the steps based on results from each step.

Another way to say it is that automation is a subset of orchestration – you cannot orchestrate manual, non-automated tasks – however multiple automation tasks strung together are not orchestration unless they include programmatic control over the process based on the results of each task.

Automation and orchestration are not meant to replace the role of the system administrator, but aim to help us in creating more reliable automated tasks, and give us time to focus on innovation, problem solving, or studying new technologies, instead of day to day manual tasks.

Figure 1-1 illustrates the differences between automation and orchestration.

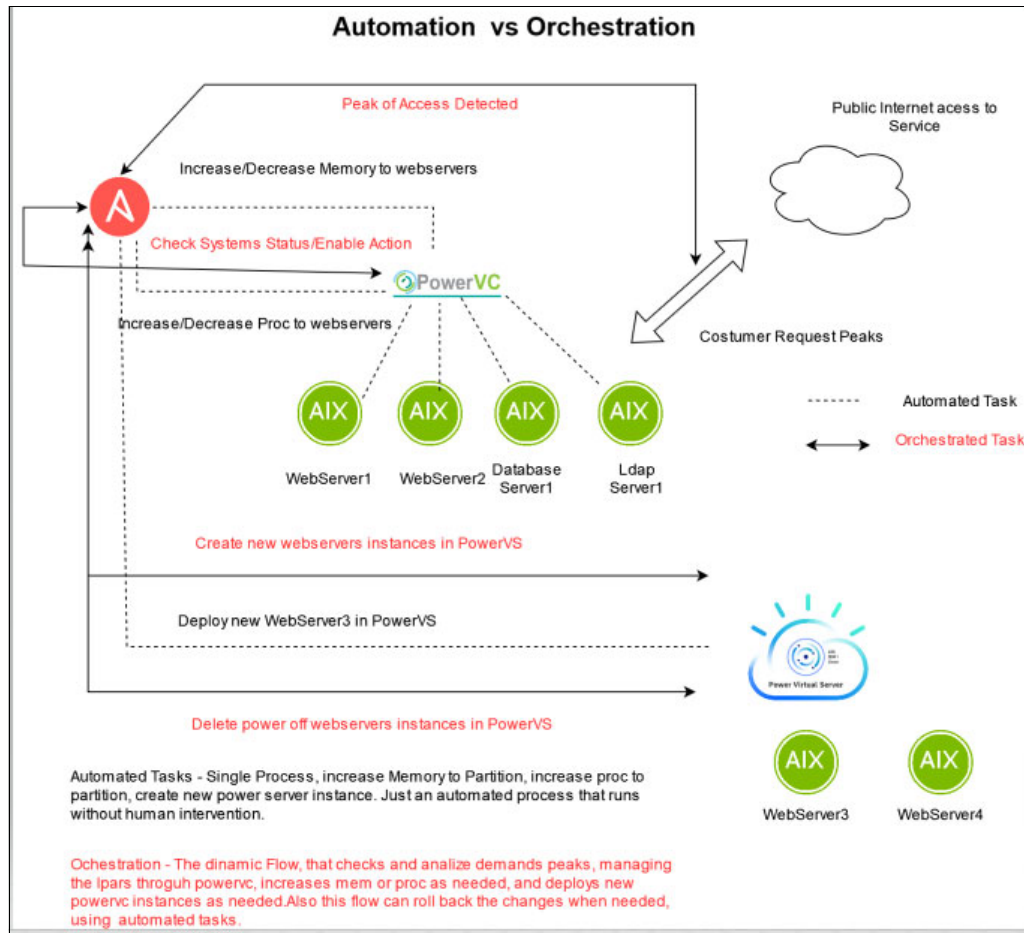


Figure 1-1 Automation versus Ochestration

1.2 Automation tools and techniques

Automation of IT processes has been a focus for many years now and a wide variety of tools and techniques have been developed that companies were using. Then 2020 came and the world almost ground to a halt due to the COVID-19 pandemic, and this accelerated the pace of automation as many companies that had not previously started thinking about automation suddenly were forced to start using automation to address the problems caused by the crisis.

1.2.1 Common IT automation tools

There are a number of tools available for automation in the IT environment. Here is a list of some of the more widely used tools:

- ▶ Chef (now Progress Chef) is a configuration management tool written in Ruby and Erlang. It uses a pure-Ruby, domain-specific language (DSL) for writing system configuration “recipes”. Chef is used to streamline the task of configuring and maintaining a company's servers and can integrate with cloud-based platforms such as Amazon EC2, Google Cloud Platform, Oracle Cloud, OpenStack, IBM Cloud, Microsoft Azure, and Rackspace to automatically provision and configure new machines.

- ▶ Puppet is an automated administrative engine for your Linux, Unix, and Windows systems, performs administrative tasks (such as adding users, installing packages, and updating server configurations) based on a centralized specification Manage and automate more infrastructure and complex workflows with reusable blocks of self-healing infrastructure as code, and Quickly deploy infrastructure to support your evolving business needs at will (and at scale) with model-driven and task-based configuration management.?
- ▶ HashiCorp Terraform is an infrastructure as code tool that lets you define both cloud and on-premises resources in human-readable configuration files that you can version, reuse, and share. You can then use a consistent workflow to provision and manage all of your infrastructure throughout its lifecycle. Terraform can manage low-level components like compute, storage, and networking resources, as well as high-level components like DNS entries and SaaS features.

In August of 2023 HashiCorp announced that future versions of Terraform will be covered by the business source license (BSL) as compared to current versions being open-source under the Mozilla Public License (MPL) v2.0 license. As a result of this announcement the Linux Foundation announced the formation of OpenTofu, an open source alternative to Terraform for code provisioning.

- ▶ Ansible is an open-source, cross-platform tool for resource provisioning automation that DevOps professionals popularly use for continuous delivery of software code by taking advantage of an “infrastructure as code” approach. Over the years, the Ansible automation platform has evolved to deliver sophisticated automation solutions for operators, administrators, and IT decision-makers across various technical disciplines. It is a leading enterprise automation solution with flourishing open-source software and the unofficial standard in IT automated systems. It operates on several Unix-like platforms and can manage systems like Unix and Microsoft architectures. It comes with descriptive language for describing system settings.

Because of the broad acceptance of the Ansible platform, its open source design, and its wide support for many devices and platforms it has started to become a dominant tool in the market. However, it is also common to use some of the other automation tools in conjunction with Ansible to do more complex automation – for example many companies use Ansible in cooperation with Terraform to provide automatic provisioning of their infrastructure.

1.3 Understanding Ansible: A Powerful Automation Tool

Ansible is a versatile open-source automation platform designed to streamline IT operations. It offers a comprehensive set of features for configuring systems, deploying software, and orchestrating complex workflows. Ansible's core strengths are its simplicity, ease of use, and focus on security and reliability.

Ansible Architecture:

As illustrated in Figure 1-2 on page 6, Ansible's architecture consists of an Ansible controller and one or more Ansible client hosts. The controller executes automation tasks and houses Ansible collections, which contain modules, plugins, and roles defining the actions Ansible can perform on client nodes.

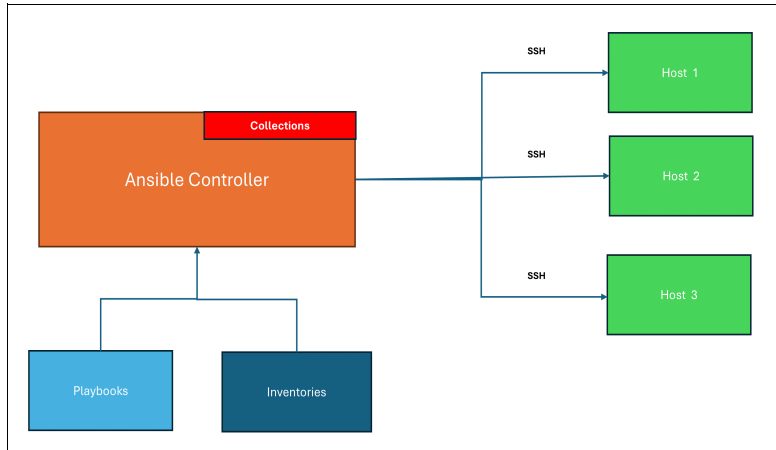


Figure 1-2 Simplified Ansible architecture

Playbooks: The Heart of Ansible Automation:

Ansible playbooks are YAML files that define sequences of tasks to be executed on remote hosts. These tasks can range from installing packages to configuring services or copying files. Playbooks enable IT teams to automate infrastructure provisioning, configuration management, application deployment, and more.

Why Choose Ansible?

Ansible offers numerous benefits for IT professionals seeking to improve efficiency, scalability, and consistency in their infrastructure. Key advantages include:

- ▶ **Versatility:** Ansible supports a wide range of devices and can scale to accommodate growing environments and automation needs.
- ▶ **Agentless Architecture:** Ansible manages devices using SSH, eliminating the need for agents on target systems.
- ▶ **Flexibility:** Ansible can be used both for simple command-line tasks and complex workflows defined in playbooks.
- ▶ **Extensive Module Library:** Ansible provides a rich collection of modules for managing various systems, cloud infrastructure, and OpenStack.
- ▶ **Declarative Approach:** Ansible's declarative syntax allows you to define the desired state of a system, and Ansible takes the necessary steps to achieve it.
- ▶ **Ease of Learning:** Ansible's simple YAML syntax and minimal learning curve make it accessible to IT professionals of all levels.

In conclusion, Ansible is a powerful and user-friendly automation tool that can help organizations improve efficiency, scalability, and reliability in their IT infrastructure. By leveraging Ansible playbooks, IT teams can streamline routine tasks, automate complex workflows, and ensure consistent configurations across their environments.

1.3.1 Options for implementing Ansible

As you decide to implement Ansible for IT management, it's essential to select the appropriate product and support level to meet your organization's needs. Here are some of the options available to you.

Ansible Community

The community versions of Ansible primarily include the following

- ▶ Ansible Core

Ansible Core is fundamental part of Ansible, providing the core automation engine. It is an open-source tool that includes the basic functionalities for configuration management, application deployment, and task automation. Ansible core includes modules, plugins, and the command-line interface needed to execute playbooks and manage configurations.

- ▶ AWX

AWX is the upstream, open-source project that serves as the community version of Red Hat Ansible Tower. AWX provides a web-based user interface, REST API, and task engine for managing Ansible automation at scale. AWX offers role-based access control, job scheduling, graphical inventory management, and more, designed to help users manage and scale automation efforts.

Note: Although AWX is free, it does not come with enterprise-level support or guarantees.

- ▶ Ansible Collections

Ansible collections are pre-packaged modules, roles, and plugins created and shared by the community. Collections allow users to extend Ansible's functionality with additional content, often maintained by the community or specific organizations. Collections can be downloaded from Ansible Galaxy, a community hub for sharing and discovering Ansible content.

- ▶ Ansible Galaxy

Ansible Galaxy is a repository for sharing and discovering Ansible roles and collections. It is a community-driven platform where users can find reusable Ansible content to simplify automation tasks. It provides a searchable repository of roles and collections created by the Ansible community, which can be integrated into your automation workflows.

These community versions are suitable for individual users, small teams, and development environments but lack the formal support and advanced features provided by Ansible Automation Platform (AAP).

Red Hat Ansible Automation Platform

Red Hat Ansible Automation Platform (RHAAP) is a subscription-based enterprise solution combining over 20 community projects into a fully supported automation platform. RHAAP provides curated, certified, and validated Ansible collections and roles from partners like IBM, Juniper, Cisco, and public cloud providers.

Key Considerations for choosing RHAAP are:

- ▶ **Support Level:** RHAAP offers enterprise-grade support, including SLAs for security, compatibility, and upgrades. Community options may have limited support.
- ▶ **Features:** RHAAP includes additional features beyond Ansible-core, such as a web interface and integration with other tools.
- ▶ **Cost:** RHAAP is a subscription-based product, while community options are free.
- ▶ **Scale and Complexity:** For large organizations with complex automation needs, RHAAP may be the better choice due to its enterprise-grade features and support.

By carefully evaluating these factors, you can select the Ansible offering that best aligns with your organization's goals, budget, and support requirements.

Table 1-1 compares these offerings.

Table 1-1 Comparison of Ansible offerings

Technology	Community/Upstream	Supported/Downstream
Ansible-core	X	
Community Ansible	X	
AWX	X	
Red Hat Ansible Automation Platform		X

Which method you use to procure Ansible is determined by your business requirements. If your automation environment is small and not business critical, it would be acceptable to use the community supported versions. However, if you are supporting business critical environments, it is important to consider the benefits of a supported enterprise product. You should consider an enterprise solution if you:

- Require enhanced security.
- Are embarking on an IT transformation initiative.
- Are ready to expand automation to include more people, teams, and use cases.
- Need flexibility to adapt to changing business requirements-with proven, innovative solutions.
- Want to prioritize automation objectives over managing automation infrastructure.

Which Ansible option is right for my organization?

Community Ansible is suitable for individuals and small teams seeking automation for personal workloads or home lab environments. For collaborative automation efforts, AWX or Ansible Automation Platform offer more robust options.

While AWX is a free, open-source project, it lacks enterprise-grade support, including SLAs for security, compatibility, and upgrade migrations. This can lead to hidden costs associated with security breaches and time-consuming fixes. However, AWX can be valuable for small-scale labs, developers contributing to the upstream code, or as a sandbox for learning Ansible Automation Platform before transitioning to an enterprise solution.

For organizations aiming to scale automation at an enterprise level, Ansible Automation Platform (AAP) is a more comprehensive choice. It offers developer tooling, flexible deployment options, and guaranteed SLAs for compatibility, upgrades, and security. AAP also provides transparent and efficient scaling of automation investments.

Table 1-2 outlines the key capabilities of each option, helping you determine whether Community Ansible, AWX, or AAP best aligns with your organization's needs.

Table 1-2 Community Ansible and AWX vs. Red Hat Ansible Automation Platform comparison

Capability	Community Ansible and AWX	Red Hat Ansible Automation Platform
Security	Not available	Trusted chain-of-custody for certified and private content.
Certified content and partner ecosystem	Not available	140+ certified content collections across 60+ partners. Benefit from prebuilt, fully supported, certified automation content from Red Hat and our partners.

Capability	Community Ansible and AWX	Red Hat Ansible Automation Platform
Life cycle support	Not available	At least 18 months of enterprise support per release. Critical bugfix and security vulnerability back porting for all components.
Legal protections	No protections	Intellectual property protections via the Open Source Assurance Agreement.
Analytics	Not available	Automation analytics and Red Hat Insights for Ansible Automation Platform offer in-depth analytics and reporting for planning and tracking performance and adoption.
Upgrades and migrations	Not supported	Supported migration to major releases as well as upgrades to minor releases.
Training and consulting	Not available	Expert resources to help you build and run a successful automation practice, backed by robust training offerings and support. Hands-on migration assistance from AWX to Ansible Automation Platform is also available.
Cloud deployment options	Not available	Managed and self-managed applications available to deploy on your cloud of choice, including Microsoft Azure, AWS, and Google Cloud. Counts toward committed spend agreements. Supported by Red Hat with integrated billing. View deployment options and pricing information.
Event-Driven Ansible	Separate upstream project that requires manual integration into your environment	Event-Driven Ansible is an integrated and tested product component of Ansible Automation Platform that reduces manual tasks, delivers more efficient IT operations, and frees your teams to focus on innovation.
Private automation hub	Separate upstream project that requires manual integration into your environment	Private automation hub is an integrated and tested product component of Ansible Automation Platform.
IBM watsonx Code Assistant for Red Hat Ansible Lightspeed	A generative AI service for task creation available to all Ansible users	A full commercial version of Ansible Lightspeed including IBM watsonx Code Assistant for Red Hat Ansible Lightspeed is available for Ansible Automation Platform subscribers.

When considering automation to address resource constraints, organizations must evaluate their readiness to manage disparate tools and their interest in contributing to open-source development models.

Ansible Automation Platform offers a distinct advantage with Event-Driven Ansible. This feature automates tasks by connecting events to corresponding actions through rules. By defining rulebooks, you enable Event-Driven Ansible to automatically recognize events, match them to appropriate actions, and execute them. This frees your teams to focus on high-value work.

Red Hat Ansible Automation Platform is a trusted enterprise solution with a proven track record. It's used by over 3,000 global customers across various industries to create, manage, and scale IT automation. As a comprehensive, integrated solution, it combines open-source innovation with enterprise-hardened features to boost productivity and accelerate project completion. We present more on Red Hat Ansible Automation Platform in the next section.

1.3.2 Red Hat Ansible Automation Platform

Red Hat Ansible Automation Platform is a robust end-to-end automation platform designed to streamline IT operations. Built on open-source innovation and hardened for enterprise environments, RHAAP offers a comprehensive set of tools for configuring systems, deploying software, and orchestrating advanced workflows. It empowers organizations to create, manage, and scale automation across their entire enterprise.

End-to-end automation

Ansible Automation Platform is the strategic automation solution. It is no longer just an upstream command line Ansible package with support, and it is not just a graphical user interface for Ansible. It is a proven enterprise platform used by every Fortune 500 company in the airline, government, and military sectors to create, manage, and scale automation strategies.¹

Create and share

Developers can write consistent code in an execution environment without constraints or operational overhead using:

- Event-Driven Ansible can automate IT actions with user-defined, rule-based constructs and create end-to-end automated scenarios for use cases across the IT landscape.
- Ansible content collections contain Red Hat Ansible Certified Content and Ansible validated content which is a reuse code to automate faster.
- Ansible content tools help developers create consistent code, streamlining the building and deployment of execution environments to speed development cycles and realize value quicker.
- Automation execution environments are container images used to execute Ansible Playbooks and roles.

Manage and measure

Operations teams can automate as intended from development through production using:

- Ansible automation hubs where can find, download, and share supported collections. And a private automation hub lets create a curated library of automation content for internal teams to share.
- Automation controller is a centralized management tool to manage inventory, launch and schedule workflows, track changes, and integrate reporting with a centralized user interface.
- Ansible Automation Platform's [trusted supply chain](#) offers enhanced security and compliance.
- Automation analytics and Red Hat Insights for Ansible Automation Platform provides rich reporting and advanced analytics to optimize your automation, proactively identify potential issues, mitigate vulnerabilities, and improve resolution times.

Scale up and out

As more people start to use the automation tools, IT architects can design and expand automation tactics across multiple environments and geographies using:

- Automation mesh, which can scale automation of large inventories across diverse network topologies, platforms, and regions.

¹ <https://www.redhat.com/en/topics/automation/why-choose-red-hat-for-automation>

Flexible design options which help you deliver automation across physical and virtual data centers, hybrid cloud environments, and edge locations, localizing automation execution to improve network performance. It helps your organization operate at a global scale.

Red Hat Ansible Automation Platform is a single automation platform for multiple use cases as below:

- **Hybrid cloud:** Automate cloud-native environments and manage infrastructure and services across public, private, and hybrid clouds with certified integrations.
- **Edge:** Standardize configuration and deployment across your entire IT landscape—from datacenter to cloud to edge environments—with a single, consistent automation platform.
- **Networks:** Manage entire network and IT processes across physical networks, software-defined networks, and cloud-based networks, all the way to edge locations.
- **Security:** Orchestrate security systems using a curated collection of modules, roles, [research from IDC](#), and playbooks to investigate and respond to threats.
- **Infrastructure:** Consistently deploy, manage, and scale infrastructure workloads where it makes the most sense in your physical datacenter, private or public cloud, or at the network edge.
- **Provisioning:** Streamline the process of PXE booting and kick starting bare-metal servers or VMs, and creating virtual or cloud instances from templates.
- **Configuration management:** Centralize configuration file management and deployment with a low learning curve for administrators, developers, and IT managers.

For more details see [RHAAP Beginners Guide Book](#).

Ansible Automation Platform core components and architecture

Ansible Automation Platform consist of several integral components. Figure 1-3 shows the Ansible Automation Platform (AAP) components and architecture².

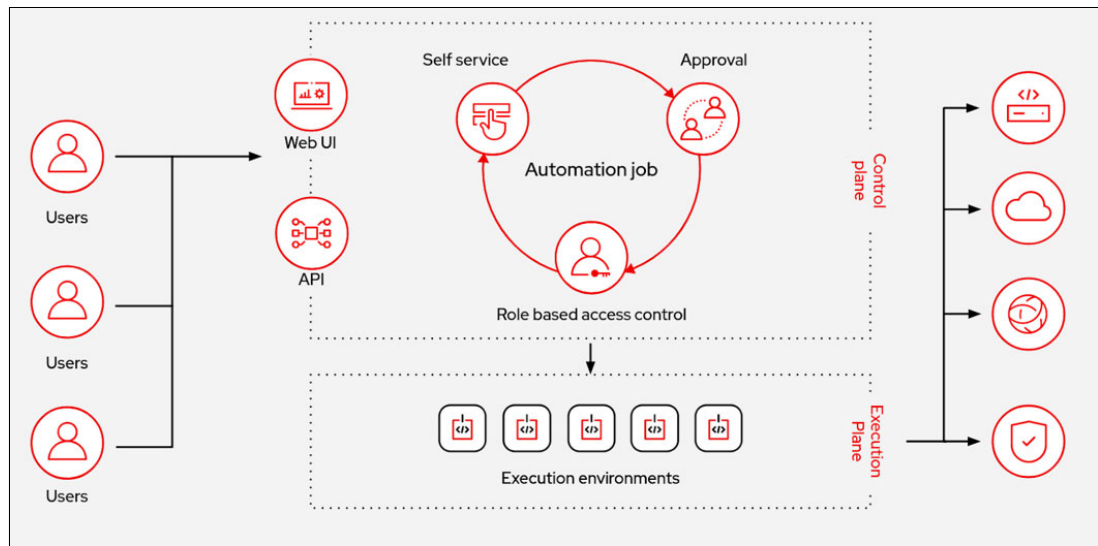


Figure 1-3 Red Hat Ansible Automation Platform core components

² <https://www.ansible.com/blog/peeling-back-the-layers-and-understanding-automation-mesh>

Automation controller

Automation controller is a distributed system, where different software components can be co-located or deployed across multiple compute nodes. Automation controller is the control plane for automation, and includes a user interface, browse-able API, role-based access control (RBAC), job scheduling, integrated notifications, graphical inventory management, CI/CD integrations, and workflow visualizer functions. Manage inventory, launch and schedule workflows, track changes, and integrate into reporting, all from a centralized user interface (UI) and RESTful application programming interface (API).

In the installer, node types of control, hybrid, execution, and hop are provided as abstractions to help the user design the topology that is appropriate for their use case. Automation hub

Automation hub enables you to discover and use new certified automation content from Red Hat Ansible and Certified Partners. On Ansible automation hub, you can discover and manage Ansible Collections, which are supported automation content developed by Red Hat and its partners for use cases such as cloud automation, network automation, and security automation.

Private automation hub provides both disconnected and on premises solutions for synchronizing collections and execution environment images from Red Hat cloud automation hub. You can also use other sources such as Ansible Galaxy or other container registries to provide content to your private automation hub. Private automation hubs can integrate into your enterprise directory and your CI/CD pipelines. Figure 1-4 shows the development cycle for an automated execution environment.

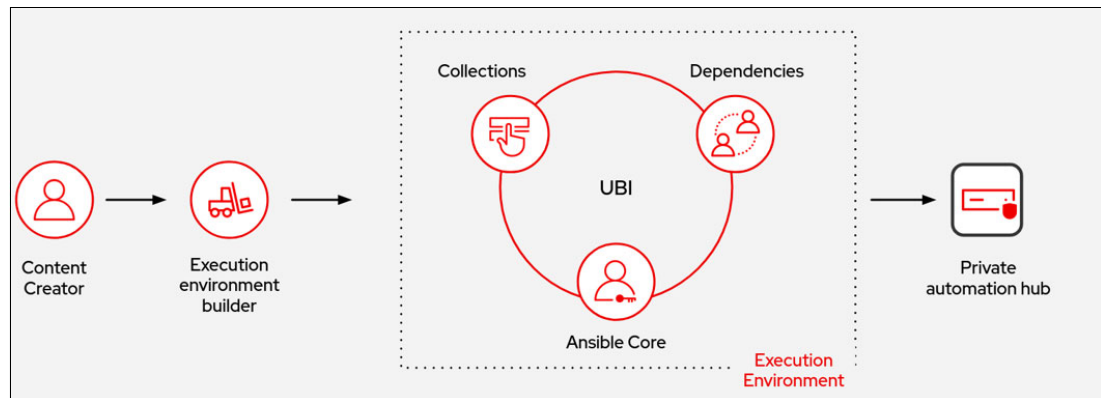


Figure 1-4 Development cycle of an automation execution environment³

Execution Environment

An automation execution environment is a container image used to execute Ansible playbooks and roles. Automation execution environments provide a defined, consistent, and portable way to build and distribute your automation environment between development and production. Execution environments give Ansible Automation Platform administrators the ability to provide and manage the right automation environments that meet the needs of different teams, such as networking and cloud teams. They also enable automation teams to define, build, and update their automation environments themselves. Execution environments provide a common language to communicate automation dependency between automation developers, architects, and platform administrators. Figure 1-5 on page 13 details the automation execution environment.

³ <https://cloudnroll.com/2023/05/22/understanding-and-creating-ansible-execution-environments/>

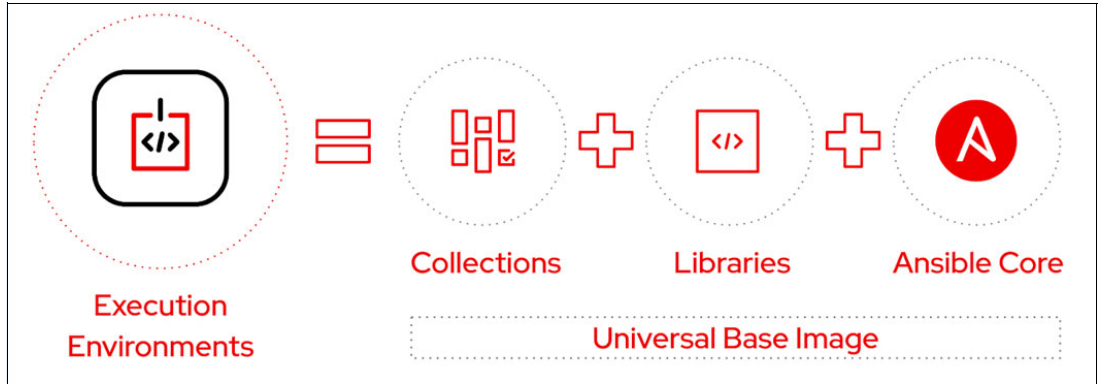


Figure 1-5 Anatomy of automation execution environment⁴

Ansible Core (Ansible Engine)

Ansible Engine is an implementation in which the strategy has slightly changed. Instead of shipping a “kitchen sink” package that is re-packaged from the upstream Ansible Project, going forward, Ansible Automation Platform will ship the ansible-core package as a standalone RPM and within execution environments.

Automation mesh

Automation mesh is an overlay network intended to ease the distribution of work across a large and dispersed collection of workers through nodes that establish peer-to-peer connections with each other using existing networks. Automation mesh makes use of unique node types to create both the control and execution plane. This is shown in Figure 1-6.

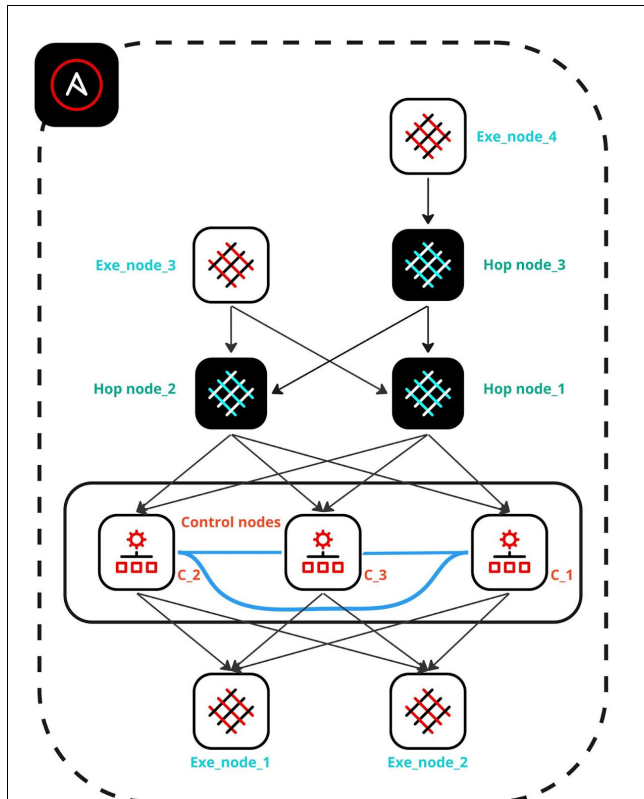


Figure 1-6 Sample automation mesh networking

⁴ <https://www.openvirtualization.pro/whats-new-in-red-hat-ansible-automation-platform-2/>

Control plane

The control plane consists of hybrid and control nodes.

- ▶ Hybrid nodes: The default node type for control plane nodes, responsible for automation controller runtime functions like project updates, management jobs and ansible-runner task operations. Hybrid nodes are also used for automation execution.
- ▶ Control nodes: The control nodes run project and inventory updates and system jobs, but not regular jobs. Execution capabilities are disabled on these nodes.

Execution Plane

The execution plane consists of execution nodes that execute automation on behalf of the control plane and have no control functions. And consists of hop and execution nodes.

- ▶ Execution nodes: run jobs under ansible-runner with podman isolation. This node type is similar to isolated nodes. This is the default node type for execution plane nodes.
- ▶ Hop nodes: Similar to a jump host, hop nodes will route traffic to other execution nodes. Hop nodes cannot execute automation.
- ▶ Peer Relationship: define the node-to-node connections between controller and execution nodes.

More information can be found in the [Red Hat Ansible Automation Platform \(RHAAP\)](#) documentation.

1.3.3 Event-Driven automation

Event-driven automation is the process of responding automatically to changing conditions in an IT environment, to help resolve issues faster and reduce routine, repetitive tasks⁵.

Event-driven automation helps connect data, analytics, and service requests to automated actions so that activities, such as responding to an outage or adjusting some aspect of an IT system that can take place in a single, rapid motion. Automating in an “if-this-then-that” fashion helps IT teams manage how and when to target specific actions. Figure 1-7 shows a typical event-driven automation environment.

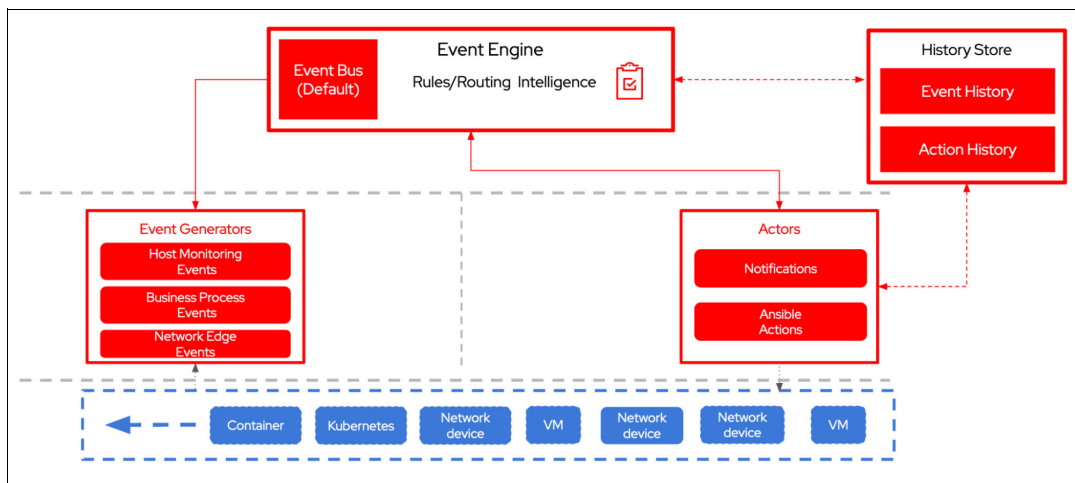


Figure 1-7 Typical event-driven environment⁶

⁵ <https://www.redhat.com/en/topics/automation/what-is-event-driven-automation>

⁶ <https://www.redhat.com/en/blog/achieving-speed-and-accuracy-through-event-driven-automation>

What is an IT event?

An event refers to any detectable occurrence that has significance for the management of IT infrastructure or the delivery of an IT service. Events are often identified by third party monitoring tools, and typically indicate significant occurrences or changes of state in applications, hardware, software, cloud instances, or other technologies.

What does it mean to be event-driven?

In an IT environment, being event-driven means connecting data and service requests to automated actions, so that manual steps typically taken by IT teams can happen in a single automated workflow. Event-driven automation allows systems to initiate a predefined automated response when an event occurs.

For example, a system outage can trigger an event that automatically executes a specific action, such as logging a trouble ticket, gathering facts needed for troubleshooting, or performing a reboot. Since these actions are predefined and automated, they can be performed more quickly than if the required steps were done manually.

What does event-driven automation offer IT teams?

As organizations strive to use automation more strategically across hybrid cloud environments and edge locations, they often start by automating IT actions that are central to management and service delivery. While automation can increase the speed and agility of these processes and minimize human error, some events still require manual troubleshooting and information gathering, which can delay resolution and disrupt everyday operations.

Event-driven automation can help teams move from a reactive to a proactive approach to IT management and streamline IT actions with full end-to-end automation. Solutions with event-handling capabilities extend the use of automation across domains, processes, and geographies, which advances automation maturity by ensuring operational consistency, resilience, and efficiency.

Event-driven automation can help IT teams to:

- ▶ Select ideal tasks to automate, then allow IT domain experts, such as a network engineer to flexibly apply automation to key needs.
- ▶ Build existing operational knowledge into automated decision-making and actions.
- ▶ Complete repetitive tasks efficiently and deliver services more quickly.
- ▶ Reduce low-level tasks and use valuable resources for other priorities.
- ▶ Address festering problems rapidly, before they become urgent issues.
- ▶ Automate repetitive tasks for networking, edge, infrastructure, DevOps, security, and cloud.

Event-driven automation use cases

Getting started with event-driven automation begins with identifying repetitive, mundane tasks that IT teams complete manually and frequently. Some common use cases include:

- ▶ **Automated remediation:** Event-driven automation can connect the analytics or tickets that flag an issue to the automated steps that will resolve it. This means that teams can automate the resolution of tickets, the remediation of issues based on known system-behavior patterns, or the response to monitored events, such as an alert that a system needs more capacity.
- ▶ **Ticket enrichment:** Event-driven automation can be used to reach out to relevant systems, gather data, and update corresponding tickets with the rich detail needed for a more thorough effective root cause analysis (RCA) process.
- ▶ **Automated platform scaling:** Application workloads and platforms rely on automated provisioning to ensure business continuity and reduce the potential impact on customers. Rather than waiting for manual provisioning, IT teams can combine capacity and

performance metrics with event-driven automation to automatically provision containers, cloud infrastructure, virtual machines, and other technologies.

- ▶ Risk mitigation: With event-driven automation, security responses can be launched as soon as a risk is identified. For example, if a risk is identified on a firewall, an event-driven solution can immediately close down the firewall and create a service ticket, reducing the opportunity for exposure to a security breach.
- ▶ Automated tuning and capacity management: Ongoing tuning and capacity management are necessary for many IT functions, such as managing web applications and monitoring storage pools. For some teams, tuning occurs thousands or tens of thousands of times per month, making it time-consuming when done manually. Event-driven automation can respond to these types of events based on predetermined rules to address things like low storage capacity-and trigger automatic adjustments.
- ▶ Scaling automation: As with tuning, it can be burdensome to manually scale applications' storage, processing, and network bandwidth to meet user demand. For example, an event-driven automation solution can monitor buffer pools, automatically adjusting sizes as limits are reached.

For more information see [what is event driven automation](#) in the Red Hat documentation.

1.3.4 Infrastructure as code - Integration of Ansible and Terraform

In the modern landscape of cloud computing and DevOps practices, Infrastructure as Code (IaC) has emerged as a pivotal concept. Two of the most prominent tools in this realm are Ansible and Terraform. While both tools contribute significantly to automating infrastructure management, they do so with distinct philosophies and purposes. In this section, we'll delve into the differences and overlaps in features between Ansible and Terraform, shedding light on when and how to best leverage each tool.

Ansible

Ansible is known for its focus on orchestration and configuration management. It excels at automating tasks that involve the setup, configuration, and management of systems, applications, and networks. Ansible uses a declarative approach, where you define the desired state of your systems, and Ansible takes care of bringing them to that state. Ansible playbooks, written in YAML, encapsulate these declarative configurations and automation workflows.

Configuration management

Ansible shines when it comes to ensuring consistency across a variety of systems. Its idempotent nature ensures that tasks are only executed if necessary, reducing the risk of unintended changes.

Ad-hoc commands

Ansible allows for quick and flexible execution of ad-hoc commands across multiple servers. This feature is particularly useful for tasks that require immediate attention or investigation.

Overlap with Terraform

While Ansible can manage provisioning tasks, it is not designed as a full-fledged infrastructure provisioning tool like Terraform. However, Ansible and Terraform can complement each other by allowing Ansible to handle complex configuration tasks after Terraform provisions the infrastructure.

Terraform

Terraform, on the other hand, is purpose-built for provisioning and managing infrastructure. It employs a declarative language to define the infrastructure's desired state, creating a clear separation between the “what” and the “how.” Terraform's configuration files, written in HashiCorp Configuration Language (HCL), describe the infrastructure resources, their dependencies, and relationships.

Infrastructure provisioning

Terraform's strength lies in its ability to create and manage infrastructure resources across various cloud providers and on-premises environments. It excels at managing the lifecycle of resources, from creation to updates and destruction.

State management

Terraform maintains a state file that records the current state of the infrastructure. This state file allows Terraform to determine what changes are necessary to reach the desired state and helps prevent accidental changes.

Overlap with Ansible

While Terraform can provision infrastructure, it is not designed for handling configuration management tasks like Ansible. However, Terraform and Ansible can work together by leveraging Ansible's capabilities to configure the provisioned infrastructure.

Complementary roles

Ansible and Terraform, are not mutually exclusive; in fact, they often work best when used in tandem. Terraform excels at setting up the infrastructure, ensuring resources are created and managed accurately, while Ansible takes charge of configuring and maintaining the systems running on that infrastructure. This synergistic approach maximizes the strengths of both tools while minimizing their respective weaknesses. Understanding that the two tools can work well together, Red Hat has created two certified collections to help you to better integrate the two tools.

The [Terraform Collection for Ansible Automation Platform](#) automates the management and provisioning of infrastructure as code using the Terraform CLI tool within Ansible playbooks and Execution Environment runtimes. The [Ansible provider for Terraform](#) allows your Terraform workflows to integrate to your Ansible workflows by collecting the build results to populate an Ansible inventory for further automation with Ansible.

In conclusion, Ansible and Terraform offer distinct yet complementary features in the world of Infrastructure as Code. While Ansible excels at orchestration and configuration management, Terraform focuses on provisioning and managing infrastructure. By understanding their differences and overlaps, DevOps practitioners can harness the power of both tools to create a robust, automated, and efficient infrastructure management.

IBM Cloud provisioning

Running your infrastructure in the cloud also benefits from Infrastructure as Code to allow for automated provisioning and management. The IBM Cloud provides a service for infrastructure provisioning of environments in the IBM Cloud utilizing both Terraform and Ansible.

IBM Cloud schematics

IBM Cloud Schematics is a managed service that offers both Terraform and Ansible in a unique way where you can provision infrastructure using Terraform and trigger Ansible configuration management in the same deployment. Most IBM Cloud resources across IBM

Cloud Classic, IBM VPC and IBM Power Virtual Server can be provisioned using Terraform and Ansible.

1.3.5 Provisioning

Automated infrastructure provisioning is the first step in automating the operational life cycle of your applications. From traditional servers to the latest serverless or function-as-a-service environments, Red Hat Ansible Automation Platform (RHAAP) can provision cloud platforms, virtualized hosts and hypervisors, applications, network devices, and bare-metal servers. It can then connect these deployed nodes to storage, add them to a load balancer, patch them for security, or perform any number of other operational tasks executed by separate teams.

Ansible Automation Platform is the single platform in your process pipeline for deploying infrastructure and connecting it, simplifying the deployment and day-to-day management of your infrastructure. Consider the following components of your infrastructure that can be provisioned by Ansible:

- ▶ **Bare metal:** Underneath virtualization and cloud platforms is bare metal, and you still need to provision it depending on the situation. Ansible Automation Platform integrates with many datacenter management tools to both invoke and enact the provisioning steps required.
- ▶ **Virtualized:** From hypervisors to virtual storage and virtual networks, you can use Ansible Automation Platform to simplify the experience of cross platform management. The large selection of available integrations gives you flexibility and choice to manage your diverse environment.
- ▶ **Networks:** Ansible's network automation capabilities allow users to configure, validate, and ensure continuous compliance for physical network devices. Ansible Automation Platform can easily provision across multi-vendor environments, often replacing manual processes.
- ▶ **Storage:** Ansible Automation Platform can provision and manage storage in your infrastructure. Whether it's software-defined storage, cloud-based storage, or even hardware storage appliances, you can find a module to benefit from Ansible's common, powerful language.
- ▶ **Public cloud:** Ansible Automation Platform is packaged with hundreds of modules supporting services on the largest public cloud platforms. Compute, storage, and networking modules allow playbooks to directly provision these services. Ansible can even act as an orchestrator of other popular provisioning tools.
- ▶ **Private cloud:** One of the easiest ways to deploy, configure, and orchestrate OpenStack private cloud is by using Ansible Automation Platform. It can be used to provision the underlying infrastructure, install services and applications, add computer hosts, and more.

Centralize your automation

Automation controllers help you scale IT automation, manage complex deployments, and speed productivity. Centralize and control your IT infrastructure with a visual dashboard, role-based access control, job scheduling, integrated notifications, and graphical inventory management. Automation controller's REST API and CLI make it easy to embed into existing tools and processes.

Figure 1-8 shows actions that can be improved by automation, as you can see it includes the full lifetime of your IT assets.

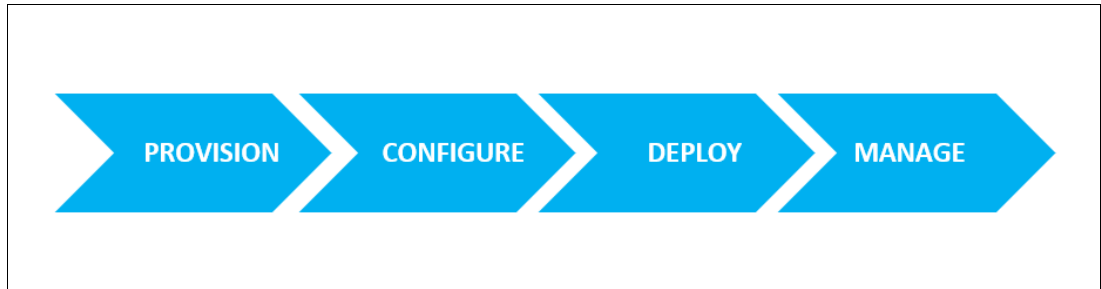


Figure 1-8 Automation doesn't stop with provisioning

Provisioning with Ansible Automation Platform allows you to seamlessly transition into configuration management, orchestration, and application deployment using the same simple, human-readable automation language. For more information on provisioning with Ansible see <https://www.redhat.com/en/technologies/management/ansible/provisioning>.

1.3.6 Patch management

One of the most critical activities with a direct relation to business continuity is the application of patches on your systems. If a critical fix is required to cover any potential breach, it is absolutely necessary to apply it as soon as possible, while at the same time ensuring that none of your systems goes offline for too long a time that it affects your running services. Hence, the need for a reliable way of executing an effective patch management strategy.

Ever wonder how you can apply patches on your systems, restart, and continue working with minimal downtime? Ansible can also function as a simple management tool to make patch management easy. Complicated administration tasks that take hours to complete can be managed easily with Ansible.

While configuration management deals with maintaining the integrity and consistency of your system's components, patch management concentrates on updating and applying patches to these components. Through the use of packaging modules and playbooks, Ansible effectively minimizes the amount of time required to patch your systems. Whenever you receive alerts for Common Vulnerabilities and Exposure (CVE) notifications or Information Assurance Vulnerability Alerts (IAVA), Ansible enables you to act swiftly in response to any potential dangers to your infrastructure.

1.3.7 Security and compliance

Security and compliance management is the ongoing process of monitoring and assessing systems to ensure they comply with industry and security standards, as well as corporate and regulatory policies and requirements.

Security and compliance management is important because noncompliance may result in fines, security breaches, loss of certification, or other damage to your business. Staying on top of compliance changes and updates prevents disruption of your business processes and saves money.

To successfully monitor and manage compliance for your business's infrastructure, you'll need to:

- Assess:** Identify systems that are non-compliant, vulnerable, or unpatched.
- Organize:** Prioritize remediation actions by effort, impact, and issue severity.
- Remediate:** Quickly and easily patch and reconfigure systems that require action.
- Report:** Validate that changes were applied and report change results.

Security and compliance management challenges

A few things that can make security and compliance management difficult are:

► Changing security and compliance landscapes

Security threats and compliance changes evolve quickly, requiring rapid response to new threats and evolving regulations.

► Distributed environments across multiple platforms

As infrastructures become more distributed across on-site and cloud platforms, it becomes more difficult to get a complete view of your environment and any risks and vulnerabilities that might be present.

► Large environments and teams

Large, complex infrastructures and teams can complicate coordination across your environment and organization. In fact, system complexity can increase the cost of a data breach.

Security and compliance best practices and recommended tools

The best way to meet each of these challenges is with a multifaceted approach that will monitor all environments, identify any regulatory inconsistencies, address those inconsistencies and bring them up to date and into compliance, and keep a record of these updates.

These best practices can help you stay abreast of any regulatory changes and keep your systems compliant:

1. Regular system scans: Daily monitoring can help you identify compliance issues, as well as security vulnerabilities, before they impact business operations or result in fees or delays.
2. Deploy automation: As the size of your infrastructure grows and changes, it becomes more challenging to manage manually. Using automation can streamline common tasks, improve consistency, and ensure regular monitoring and reporting, which then frees you up to focus on other aspects of your business.
3. Consistent patching and patch testing: Keeping systems up to date can boost security, reliability, performance, and compliance. Patches should be applied once a month to keep pace with important issues, and patching can be automated. Patches for critical bugs and defects should be applied as soon as possible. Be sure to test patched systems for acceptance before placing them back into production.
4. Connect your tools: Distributed environments often contain different management tools for each platform. Integrate these tools via application programming interfaces (APIs). This allows you to use your preferred interfaces to perform tasks in other tools. Using a smaller number of interfaces streamlines operations and improves visibility into the security and compliance status of all systems in your environment.

Some of the security and compliance tools that can help are:

- ▶ **Proactive scanning:** Automated scanning can ensure systems are monitored at regular intervals and alert you to issues without expending much staff time and effort.
- ▶ **Actionable insight:** Information that is tailored to your environment can help you more quickly identify which compliance issues and security vulnerabilities are present, which systems are affected, and what potential impacts you can expect.
- ▶ **Customizable results:** Define business context to reduce false positives, manage business risk and provide a more realistic view of your security and compliance status are ideal.
- ▶ **Prescriptive, prioritized remediation:** Prescriptive remediation instructions eliminate the need to research actions yourself, saving time and reducing the risk of mistakes. Prioritization of actions based on potential impact and systems affected help you make the most of limited patching windows.
- ▶ **Intuitive reporting:** Generating clear, intuitive reports about which systems are patched, which need patching, and which are non-compliant with security and regulatory policies increases auditability and helps you gain a better understanding of the status of your environment.

Compliance and automation with IBM - Red Hat

An automation strategy goes a long way to building capacity for checking systems for compliance without increasing time or cost. Manual compliance practices are more time-consuming, prone to human error, and harder to repeat or verify.

Selecting the right automation technologies is key for rapid implementation across the data center and network software systems in hybrid environments.

Red Hat

Red Hat has an end-to-end software stack for automation and management that includes:

- Red Hat Enterprise Linux
- Red Hat Ansible Automation
- Red Hat Satellite
- Red Hat Insights

IBM

IBM has:

- IBM QRadar®
- IBM PowerSC
- IBM Instana™
- IBM Turbonomic®

1.3.8 Configuration management

In today's IT environment, there is a growing number of servers/containers running to meet your business requirements. As the number of these instances increase, the amount of time spent managing and validating the configuration of those instances is also increasing. Ansible simplifies and automates configuration management, changing system parameters and devices for example. This allows you to maintain a consistent setup across your enterprise, even in large environments with a large number of systems.

An important aspect of configuration management is that you need to be assured that when you run a Playbook, that you will get the results you expect, that the resultant configuration

will match what is defined in the Playbook. Ansible handles this by ensuring that Playbooks are idempotent in nature. Idempotent is defined as being able to run a Playbook over and over again with the results being the same each time that Playbook is executed. This ensures that when you run a Playbook to change a configuration parameter, the resulting configuration will match what is defined in the Playbook.

1.3.9 Business continuity

Business continuity is an important aspect to keeping your infrastructure running in order to meet the demands of your users. Business continuity can be thought of a combination of technologies that provide a highly available environment, but it also involves the planning of your infrastructure to continue running in the event of a disaster at the location of your primary data center, for example, a power outage, a fire or a natural disaster resulting in a flood or other damage. Ensuring that your infrastructure is set up correctly to handle any hardware or software failures that occur and site wide outages can be complex.

Ansible is a valuable tool for enhancing business continuity and disaster recovery (BCDR) efforts by automating various tasks and processes related to system recovery, data backup, and infrastructure provisioning. Using Ansible, new servers and instances can be created, either in the same site for high availability, or in another site for disaster recovery. ensuring the infrastructure is ready when needed. In the event of increased demand during a disaster, Ansible helps to scale resources dynamically to handle the load and maintain business operations.

1.3.10 Application development

Modern business relies on applications. Not only are applications essential for your business, but they provide a competitive advantage. Fast, reliable application development is critical for success in a digital world.

Continuous integration/continuous deployment (CI/CD) approaches can help you rapidly build, test, and deliver high-quality applications. CI/CD applies automation throughout the application life cycle – from the integration and testing phases to delivery and deployment – to quickly produce tested, verified applications. It incorporates two different but related functions:

- Continuous integration (CI) helps developers rapidly verify functionality and merge their code changes back to a shared branch more frequently. Merged code changes are validated by automatically building the application and running different levels of automated testing—typically unit and integration tests—to ensure the changes work. If testing discovers a conflict between new and existing code, CI makes it easier and faster to fix those bugs.
- Continuous deployment (CD) automates the process of releasing an application to production. There are few manual gates in the development pipeline stage just before production, so CD relies heavily on well-designed test automation. As a result, a developer's change to a cloud application could go live within minutes of writing it if it passes all automated tests. CD makes it much easier to continuously receive and incorporate user feedback. Together, CI and CD practices allow you to release changes to applications in smaller pieces, making application deployment more reliable.

You can apply CI/CD to many components and assets within your organization, including applications, platforms, infrastructure, networking, and automation code. Automation is at the core of CI/CD pipelines. By definition, CI/CD pipelines require automation. While it is possible to manually execute each step in your development workflow, automation maximizes the

value of your CI/CD pipeline. It ensures consistency across development, test, and production environments and processes, allowing you to build more reliable pipelines.

Ansible automates the major stages of continuous integration, delivery, and deployment (CI/CD) pipelines – becoming the activating tool of DevOps methodologies. The automation technology you choose can affect the effectiveness of your pipeline. Ideal automation technologies include these key features and capabilities:

- Ansible offers a simple solution for deploying applications. It gives you the power to deploy multi-tier applications reliably and consistently, all from one common framework. You can configure key services as well as push application files from a single common system.
- Rather than writing custom code to automate your systems, your team writes simple task descriptions that even the newest team member can understand – saving not only up front costs, but making it easier to react to change over time.

Ansible allows you to write playbooks that describe the desired state of your systems, and then it does the hard work of getting your systems to the desired state. Playbooks make your installations, upgrades, and day-to-day management repeatable and reliable.

1.4 Introduction to IBM Power

IBM Power is a family of midrange systems that are capable of running mission critical workloads utilizing hybrid multicloud technologies. IBM Power servers are high-performance, secure, and reliable servers built on IBM's Power processor architecture. Figure 1-9 is a view of one of the newest systems – the IBM Power E1080 – in the IBM Power product portfolio.



Figure 1-9 View of Power E1080 node

IBM Power is built to be scalable and powerful, while also providing flexible virtualization and management features. IBM Power supports a wide range of open-source tools, including Ansible.

Many of the most mission-critical enterprise workloads are run on IBM Power. The core of the global IT infrastructure, encompassing the financial, retail, government, health care, and every other sector in between, is comprised of IBM Power systems, which are renowned for their industry-leading security, reliability, and performance attributes. For enterprise applications, including databases, application and web servers, ERP, and AI many clients use IBM Power.

Enterprise IT delivery is changing as a result of digital transformation, and cloud computing is playing a key role. When it comes to consuming infrastructure, you need options and

flexibility, and IBM Power are completely prepared for the cloud. Whether you're using Kubernetes and Red Hat OpenShift to modernize enterprise applications, creating a private cloud environment within your data center with adaptable pay-as-you-go services, using IBM Cloud to launch applications as needed, or creating a seamless hybrid management experience across your multicloud landscape, IBM Power delivers whatever hybrid multicloud approach you choose.

The modern data center consists of a combination of on-premises and off-premises, multiple platforms, such as IBM Power, IBM Z®, and x86. The applications range from monolithic to cloud-native – inherently some combination of bare metal, virtual machines, and containers. An effective hybrid cloud management solution must account for all of these factors. IBM and Red Hat are uniquely positioned to best accommodate the applications that you're running today and the modernized applications of tomorrow, wherever they are.

IBM Power delivers one of the highest availability ratings among servers⁷

IBM Power delivers 99.999% availability, giving 25% less downtime than comparable offerings, due to built-in recovery and self-healing functions for redundant components. Organizations are also able to switch from an earlier Power server to the current generation while applications continue to run, giving you high-availability and minimal downtime when migrating.

IBM Power is consistently rated as one of the most secure systems in the market⁸

For the fourth straight year, IBM Power has been rated as one of the most secure systems in 2022, with only 2.7 minutes or less of unplanned outages due to security issues. This puts IBM Power:

- ▶ 2x more secure than comparable HPE Superdome servers,
- ▶ 6x compared to Cisco UCS servers,
- ▶ 16x compared to Dell PowerEdge servers,
- ▶ 20x compared to Oracle x86 servers,
- ▶ and up to more than 60x compared to unbranded white box servers.

Security breaches were also detected immediately or within the first 10 minutes in 95% of the IBM Power systems that were surveyed. This results in better chances that a business will suffer little to no downtime, nor will they be susceptible to damaged, compromised, or stolen data.

IBM Power allows businesses to boost operational efficiency to meet sustainability goals⁹

A recent user case study illustrated how IBM Power enabled a customer to increase end-user application performance by 20%, ending up with them meeting their sustainability goals. By helping move the customer to IBM Power and IBM FlashSystem® storage, they were fully able to leverage their SAP S/4HANA operations enabling them to meet their climate objectives.

IBM Power streamlines AI operations with advanced on-chip technologies¹⁰

IBM Power systems delivers 5X faster AI inferencing per socket for high precision math over the previous generation. This is accomplished through multiple Matrix Math Accelerator (MMA) units in each Power processor core. MMAs allow IBM Power systems to forgo external accelerators, such as GPUs and related device management, when running machine learning and inferencing workloads.

⁷ [ITIC 2022 Global Server Hardware, Server OS Reliability Report](#)

⁸ [ITIC 2022 Global Server hardware, Server OS Security](#)

⁹ <https://www.ibm.com/case-studies/mondi-group-systems-hardware-sap-s4-hana>

Current IBM Power systems can range from scale-out servers that start with 4 cores and 32GB of memory on the IBM Power S1014 to enterprise systems with up to 240 cores and 64TB of memory on the IBM Power E1080.

Note: IBM's full lineup of server models based on the latest Power processors can be found [here](#).

1.4.1 Power processors and architecture

The Power processor is a family of 64-bit superscalar, simultaneous multithreading, multicore microprocessors designed and sold by IBM. Power processors are mainly used for IBM's Power line of servers, the Hardware Management Console (HMC), and also in IBM's storage solutions such as the DS8900F and the IBM Converged Archive Solution.

As an architecture, Power-based processors have been used in various applications such as network routers, workstations, game consoles, and even on the Curiosity and Perseverance rovers on Mars.

As AI infrastructure challenges increase due to more AI models being deployed in production, IBM Power addresses these challenges with in-core AI inferencing and machine learning through its built-in Matrix Math Accelerator (MMA). This allows you the capability of "AI at the point data" where you can perform AI inferencing without needing to ingest your data from an external source. This gives AI operations a significant performance boost.

The Power processor also provides security within the system itself through Transparent Memory Encryption, where data is encrypted by cryptography engines located in the processor core, right where memory is located. This gives you four times the speed than average encryption.

Power also features reliability, availability, and serviceability (RAS) capabilities such as advanced recovery, diagnostics, and Open Memory Interface (OMI) attached advanced memory DIMMs that deliver 2X better reliability and availability than industry standard DIMMs.

The latest version of Power processors is the Power10, built on a 7nm design that is 50% faster than its predecessor and 33% more energy efficient.

Power10 chip benefits are the result of important evolutions of many of the components that were in previous IBM POWER® chips. Several of these important Power10 processor improvements are listed in Table 1-3.

Table 1-3 Power10 processor chip technology

Technology	Power10 processor chip
Processors die size	602 mm ²
Fabrication technology	<ul style="list-style-type: none"> ▶ CMOSa 7-nm lithography ▶ 18 layers of metal
Maximum processor cores per chip	15
Maximum execution threads per core / chip	8 / 120
Maximum L2 cache core	2 MB

¹⁰ <https://www.ibm.com/it-infrastructure/resources/power-performance/e1080/>

Technology	Power10 processor chip
Maximum On-chip L3 cache per core / chip	8 MB / 120 MB
Number of transistors	18 billion
Processor compatibility modes	Support for Power Instruction Set Architecture (ISA) of POWER8 and POWER9

Figure 1-10 shows the Power10 processor die with several functional units that are labeled. Sixteen SMT8 processor cores are shown, but the dual-chip module (DCM) with two Power10 processors provide 12-, 18-, or 24-core for Power E1050 server configurations.

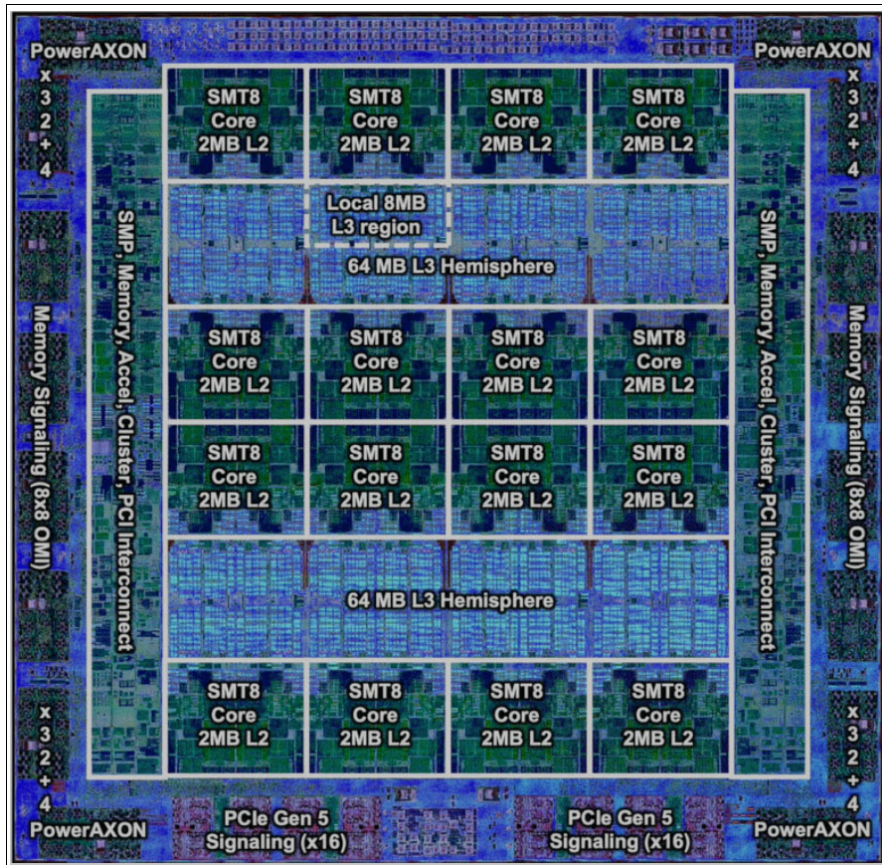


Figure 1-10 The Power10 processor chip (die photo courtesy of Samsung Foundry)

1.4.2 PowerVM and virtualization

IBM PowerVM is a virtualization environment feature available on IBM Power that provides support for virtual machines (VMs) by enabling the creation of micro-partitions (also known as logical partitions or LPARs). PowerVM allows you to consolidate VMs running multiple workloads onto fewer systems, resulting in reduced costs, increased efficiency, better return on investment, faster deployment, workload security, and better server utilization. PowerVM enables a Power server to have up to 1000 virtual machines on a single server running a mix of various operating systems and environments simultaneously.

PowerVM also provides IBM Power other advanced features, including, but not limited to:

▶ **Micro-partitioning**

Allows a partition/VM to initially occupy as small as 0.05 processing units, or one-twentieth of a single processor core, and allows adjustments as small as a hundredth (0.001) of a processor core. This allows tremendous flexibility in the ability to adjust your resources according to the exact needs of your workload.

▶ **Shared Processor Pools**

Allows for effective overall utilization of system resources by automatically applying only the required amount of processor resource needed by each partition. The hypervisor can automatically and continually adjust the amount of processing capacity allocated to each partition/VM based on system demand. You can set a shared processor partition so that, if a VM requires more processing capacity than its assigned number of processing units, the VM can use unused processing units from the shared processor pool.

▶ **Virtual I/O Server (VIOS)**

VIOS provides the ability to share storage and network resources across several VMs simultaneously, thereby avoiding excessive costs by configuring the precise amount of hardware resources needed by the system.

▶ **Live Partition Mobility (LPM)**

LPM brings the ability to move running VMs across different physical systems without disrupting the operating system and applications running within them.

▶ **Share Storage Pools (SSP)**

SSP provides the ability to provide distributed storage resources to VIO servers in a cluster.

▶ **Dynamic LPAR operations (DLPAR)**

Introduces the ability to dynamically allocate additional resources, such as available cores and memory, to a VM without stopping the application.

▶ **Performance and Capacity Monitoring**

Supports gathering of important statistics to provide an administrator information regarding physical resource distribution among VMs and continuous monitoring of resource utilization levels, ensuring they are evenly distributed and optimally used.

▶ **Remote Restart**

Allows for quick recovery in your environment by allowing you to restart a VM on a different physical server when an error causes an outage.

Note: For a more comprehensive description of PowerVM and its capabilities, you may refer to the IBM Redbooks publication *Introduction to IBM PowerVM*, SG24-8535.

1.4.3 Supported operating systems

As of the time of this publication, Power10 processor-based systems are supported by the operating system versions shown in Table 1-4 on page 28.

Table 1-4 Power10 operating system support

Operating System	Supported versions
AIX	7.3 TL0 or later 7.2 TL4 or later (with any I/O configuration) 7.1 TL5 or later (through VIOS only)
IBM i	7.3 TR12 or later (various levels of 7.3, 7.4 and 7.5 supported)
PowerVM Virtual I/O Server	4.1.0.10 or later 3.1.4.10 or later 3.1.3.21 or later 3.1.2.40 or later
Red Hat OpenShift Container Platform	4.9 or later
Red Hat Enterprise Linux	9.0 or later 8.4 or later
SUSE Linux Enterprise Server	15.3 or later
Ubuntu	22.04 or later

Note: Software maps detailing which operating system versions are supported on which specific IBM Power server models (including previous generations of IBM Power) can be found in these [IBM Support pages](#).

1.4.4 Key benefits of IBM Power compared to x86 servers

Oftentimes, the misapprehension surrounding standardization on x86 is that it is the platform with lower acquisition costs. This is often at the expense of performance, scalability, reliability, and manageability. The return on investment and total cost of ownership of x86 servers also pale in comparison to those of the IBM Power systems. IBM delivers the better overall platform compared to x86 due to the following benefits of IBM Power:

- ▶ World record SAP SD-two tier benchmark results with 8 sockets (120 cores), beating the best 16 socket (448 cores) result from the x86 platform.¹¹
- ▶ Power delivers per-core performance that is 2.5X faster than Intel Xeon Platinum, setting a world record 8-socket single server result on the SPEC CPU 2017 benchmark.¹²
- ▶ When running containerized applications and databases on an IBM Power E1080 compared to running the same workloads on an x86 server, IBM Power delivers 48% lower 3-year TCO, 4.3X more throughput per core, and 4.1X better price-performance. This means you can run the same amount of workloads with fewer servers, four times less the footprint, four times fewer software licenses, and four times the energy savings.¹³
- ▶ For the 14th straight year, IBM Power delivers the top reliability results, better than any Intel x86 platform, and only exceeded by the IBM Z. IBM Power also reported less number of data breaches (one) in the same period compared to x86 platforms.¹⁴

¹¹ <https://www.sap.com/about/benchmark.html>

¹² <https://www.spec.org/cpu2017/results/>

¹³ <https://www.ibm.com/it-infrastructure/resources/power-performance/e1080/#5>

¹⁴ <https://itic-corp.com/itic-2022-global-server-reliability-results/>

- ▶ As shown by the list of supported operating systems in Table 1-4 on page 28, IBM Power delivers the ability to run a wide variety of AIX, IBM i, or Linux workloads simultaneously, giving you flexibility in virtualization that is unmatched by any x86 offering.

Both Power and x86 architectures are established, mature foundations for modern workloads. But Power stands out for its efficiency, deeply integrated virtualization, highly dependable availability and reliability, and its unparalleled capability of supporting enterprise-class workloads that do not require the massive infrastructure that you would need to do the same with x86 hardware.

1.5 Ansible for Power

This section discusses Ansible clients across both on-premise IBM Power environments, and IBM Cloud Power Virtual Server. Ansible is able to manage a number of clients running on IBM Power, including AIX, IBM i, Linux, HMC, VIOS and Red Hat OpenShift Container Platform. In section (3.4 Preparing your system for Ansible) we discuss setting your VMs up to be Ansible clients. As well as the generic modules and collections that Ansible provides, IBM has developed a number of collections specifically for IBM Power servers. These can be found on Ansible Galaxy for community supported content, or Red Hat Automation Hub for certified content.

IBM Power Collections on Ansible Galaxy

Ansible Galaxy is a source for community based Ansible content, which holds reusable collections from thousands of contributors, including IBM. These collections include modules, plugins, playbooks and roles that anyone can download and use.

The IBM Power collections, developed by IBM are available at the links shown in Table 1-5.

Table 1-5 Links to IBM Power collections

Collection	URL
Power AIX	https://galaxy.ansible.com/ui/repo/published/ibm/power_aix/
Power IBM i	https://galaxy.ansible.com/ui/repo/published/ibm/power_ibmi/
Power HMC	https://galaxy.ansible.com/ui/repo/published/ibm/power_hmc/
Power VIOS	https://galaxy.ansible.com/ui/repo/published/ibm/power_vios/
PowerVC	Ansible support for PowerVC is provided using the Ansible Collection for OpenStack.

IBM Power Collections on Red Hat Automation Hub

The same collections can also be downloaded from [Red Hat Automation Hub](#). Ansible automation hub features Red Hat Ansible Certified Content—prebuilt, fully supported, and certified automation content from Red Hat and many of our 60+ partners, including Cisco, Microsoft, CyberArk, Dynatrace, and ServiceNow. It also includes validated content—prebuilt playbooks and roles that you can use, customize, and learn from.

The Red Hat Automation Hub interface showing these collections is shown in Figure 1-11.

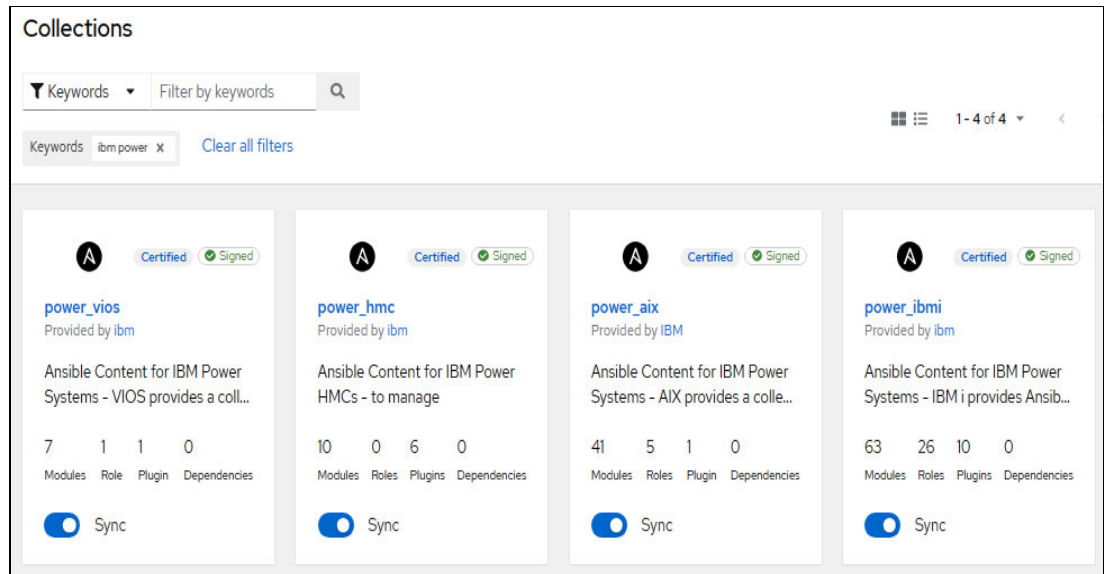


Figure 1-11 Collection on Red Hat Automation hub

The following sections discuss the benefit of using Ansible across these different environments, as well as providing detail on the specific collection contents.

1.5.1 Ansible for Linux on Power

Introduction

A major benefit of Linux is that it is open source. The software is unencumbered by licensing fees and its source code is freely available. A wide variety of Linux distributions are available for almost every computing platform.

The supported Linux distributions on IBM Power Systems are:

- ▶ Red Hat Enterprise Linux
- ▶ SUSE Linux Enterprise Server
- ▶ Ubuntu (support is available from Canonical)
- ▶ Red Hat OpenShift Container Platform

Red Hat Enterprise Linux

Red Hat Enterprise Linux (RHEL) is the most-deployed commercial Linux distribution in the public cloud¹⁵. RHEL is an open source operating system (OS) that provides a foundation to scale existing applications and deploy emerging technologies across the following environments:

- ▶ Bare metal
- ▶ Virtual
- ▶ Container
- ▶ Cloud

¹⁵ Management Insight Technologies. "The State of Linux in the Public Cloud for Enterprises," February 2018. <https://www.redhat.com/en/resources/state-of-linux-public-cloud-solutions-ebook>

SUSE Linux Enterprise Server

SLES is an enterprise Linux distribution enabled for the cloud, which also has a strong presence on the IBM Power System platform.

Ubuntu

IBM and Canonical have collaborated to make the Ubuntu distribution on IBM Power Systems. The availability of Ubuntu strengthens the Linux operating system choices available on IBM Power technology. Support for Ubuntu is available directly from Canonical.

Red Hat OpenShift Container Platform

Red Hat OpenShift Container Platform is a consistent hybrid cloud foundation for building and scaling containerized applications. Red Hat OpenShift is trusted by thousands of customers in every industry to deliver business-critical applications, whether they're migrating existing workloads to the cloud or building new experiences for customers. It is also backed by one of the leading Kubernetes contributors, Red Hat.

Red Hat OpenShift Container Platform runs on Red Hat Enterprise Linux CoreOS (RHCOS) which represents the next generation of single-purpose container operating system technology by providing the quality standards of Red Hat Enterprise Linux (RHEL) with automated, remote upgrade features. RHCOS is the only supported operating system for OpenShift Container Platform control plane (master) nodes. While RHCOS is the default operating system for all cluster machines, you can create compute (worker) nodes that use RHEL as their operating system.

Ansible supports Red Hat OpenShift Container Platform as either a control node or a client node.

Getting started with Linux management

Managing Linux on IBM Power Systems with Ansible doesn't require optional Ansible modules or collections. It is well served by the existing *ansible-core* modules included with all Ansible installations (*ansible.builtin.**) and common community modules (*ansible.community.**), some of which are shown in Table 1-6.

Table 1-6 Ansible modules commonly used with Linux targets

Ansible Module Name	Fully Qualified Collection Name	Description
apt	<i>ansible.builtin.apt</i>	Manages apt packages for Ubuntu and Debian distributions
blockinfile	<i>ansible.builtin.blockinfile</i>	This module will insert/update/remove a block of multi-line text surrounded by customizable marker lines
command	<i>ansible.builtin.command.</i>	Execute commands on targets
copy	<i>ansible.builtin.copy</i>	Copy files to remote locations
debug	<i>ansible.builtin.debug</i>	Print statements during execution
dnf	<i>ansible.builtin.dnf</i>	Manages packages with the dnf package manager. For Python 3
fetch	<i>ansible.builtin.fetch</i>	Fetch files from remote nodes

Ansible Module Name	Fully Qualified Collection Name	Description
file	<i>ansible.builtin.file</i>	Manage files and file properties
lineinfile	<i>ansible.builtin.lineinfile</i>	Manage lines in text files
template	<i>ansible.builtin.template</i>	Template a file out to a target host using <i>jinja2</i>
user	<i>ansible.builtin.user</i>	Manage user accounts
yum	<i>ansible.builtin.yum</i>	Manages packages with the yum package manager. Python 2 only.
zypper	<i>ansible.general.zypper</i>	Manage packages on SUSE

Note: Table 1-6 on page 31 just represents a small selection of Ansible modules available. A much more comprehensive list of Ansible modules and collections is described in [this document](#).

Ansible accesses Linux hosts in the inventory via SSH from the Ansible controller node, with authentication via either password, or public and private key pairs. We discuss how to choose a controller node and install Ansible in Chapter 3, “Getting started with Ansible” on page 99.

No special modules or collections are required to access a Linux host. For example, to access a Linux host named *sles15sp5* from a controller node using a user name and password, it is as simple as shown in Example 1-1.

Example 1-1 Access a Linux host using a password

```
$ ansible -m ping sles15sp5 --ask-pass --user ansible
SSH password:
sles15sp5 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3.6"
  },
  "changed": false,
  "ping": "pong"
}
```

Alternatively, to setup password-less authentication using SSH public and private keys, you can use the ‘ssh-copy-id’ command, as shown in Example 1-2, to copy a preexisting public key to a Linux host.

Example 1-2 Configure password-less login using ‘ssh-copy-id’ command

```
$ ssh-copy-id ansible@sles15sp5
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter
out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted
now it is to install the new keys
(ansible@sles15sp5) Password:
```

Number of key(s) added: 1

Now try logging into the machine, with: `"ssh 'ansible@sles15sp5'"`
and check to make sure that only the key(s) you wanted were added.

```
$ ssh ansible@sles15sp5
Last login: Mon Aug 7 16:36:31 2023 from 192.168.115.249
ansible@sles15sp5:~>
```

Once password-less logins to your Linux hosts are configured, access via Ansible becomes simpler, as shown in Example 1-3.

Example 1-3 Using password less login

```
$ ansible -m ping sles15sp5 --user ansible
sles15sp5 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3.6"
  },
  "changed": false,
  "ping": "pong"
}
```

Your Ansible command can be further simplified, by specifying the `'ansible_user'` in either the inventory or a `ansible.cfg` file.

Login options

A lot of actions you will want Ansible to perform will require superuser, or root access. There are several options that will allow you to acquire the right authorization for those operations.

Using root userid

Direct access to a Linux host as the root user is possible, and can be permitted or denied by the `'PermitRootLogin'` setting in `/etc/ssh/sshd_config` on the target host.

Valid options for `'PermitRootLogin'` are:

- ▶ `yes`
- ▶ `no`
- ▶ `prohibit-password`
- ▶ `without-password`
- ▶ `forced-commands-only`

A value of `'yes'` will allow root logins using a password, while a setting of `'no'` will deny access. The settings `'prohibit-password'` and `'without-password'` will allow root logins, but only by SSH key-pair, and not password. The option `'forced-commands-only'` will allow password-less login by root, but only to run predefined commands.

Your organization's security policy may not permit direct logins as the root user via SSH. In this case you will need to login as a regular user and then use the `'sudo'` or `'su'` command to perform tasks that requires superuser access, also called privilege escalation.

Using sudo

To use sudo, the user issuing the sudo command must be listed in the `/etc/sudoers` file on the target host. The operating system groups 'sudo' or 'wheel' are commonly used for this type of authorization. An example of a `/etc/sudoers` file allowing the members of the 'sudo' group to run any command as root is show in Example 1-4.

Example 1-4 Sample of /etc/sudoers file allowing root access for members of sudo group

```
# Allow members of group sudo to execute any command
%sudo ALL=(ALL:ALL) ALL
```

Using su

The 'su' command is another way to gain superuser privileges. In this case the user provides the root password to obtain a privilege escalation, rather than their own password. The use of 'sudo' over 'su' is preferred due to greater granularity of control via sudo.

Privilege escalation

To utilize privilege escalation in Ansible, you use the 'become' keyword. You can use the become keyword in a playbook, an `ansible.cfg` file or on the command line.

Note: Privilege escalation using become is not limited to the root user. You can specify another user to become with the 'become_user' option. For example, you might use 'become_user: apache' to perform tasks as the web server owner

A simple Ansible playbook

A common task in Ansible is to install an Operating System package, for example a tool like 'tcpdump'.

Red Hat Enterprise Linux, SUSE Linux Enterprise Server and Ubuntu all use their own separate package management systems, and in turn have separate Ansible modules to install packages.

- Red Hat Enterprise Linux: *ansible.builtin.yum*
- SUSE Linux Enterprise Server: *community.general.zypper*
- Ubuntu: *ansible.builtin.apt*

However, there is also a generic package management module named 'ansible.builtin.package' that can be used to make your playbook more portable across Linux distributions.

So based on what we have discussed so far in this section, we write our simple playbook to do the following:

- ▶ Execute our play book on the hosts in the 'power-linux' inventory group
- ▶ Login to the Linux hosts as the user 'ansible'
- ▶ Use 'sudo' to become the 'root' user
- ▶ Install the 'tcpdump' package using the generic 'ansible.builtin.package' module
- ▶ The package module will ensure the package is installed if not already present, by using the in 'state' keyword with the value of 'present'.

The playbook is shown in Example 1-5.

Example 1-5 Simple playbook install_pkg.yaml

```
---
- hosts: power-linux
  remote_user: ansible
  become: true

  tasks:
    - name: install tcpdump package
      ansible.builtin.package:
        name: tcpdump
        state: present
```

Our inventory file contains the host definitions shown in Example 1-6.

Example 1-6 Inventory example for our first playbook

```
[power-linux]
rhel7ppc64
rhel8ppc64
rhel9ppc64
sles15ppc64
ubuntu20ppc64
ubuntu22ppc64
```

To execute the playbook, we would execute the command in Example 1-7.

Example 1-7 Execute ansible-playbook command

```
ansible-playbook --ask-pass --ask-become-pass --inventory inventory install_pkg.yaml
```

The output of the ansible-playbook command is shown in Example 1-8.

Example 1-8 Output from executing the playbook

```
$ ansible-playbook --ask-pass --ask-become-pass --inventory inventory install_pkg.yaml
SSH password:
BECOME password[defaults to SSH password]:

PLAY [power-linux]
*****
*****

TASK [Gathering Facts]
*****
*****
ok: [rhel7ppc64]
ok: [ubuntu22ppc64]
ok: [rhel8ppc64]
ok: [rhel9ppc64]
ok: [ubuntu20ppc64]
ok: [sles15ppc64]

TASK [install package]
*****
*****
ok: [ubuntu20ppc64]
```

```

changed: [ubuntu22ppc64]
ok: [rhel7ppc64]
ok: [rhel9ppc64]
changed: [rhe18ppc64]
changed: [sles15ppc64]

```

PLAY RECAP

```

*****
*****
rhel7ppc64      : ok=2   changed=0   unreachable=0   failed=0   skipped=0
rescued=0      ignored=0
rhel8ppc64      : ok=2   changed=1   unreachable=0   failed=0   skipped=0
rescued=0      ignored=0
rhel9ppc64      : ok=2   changed=0   unreachable=0   failed=0   skipped=0
rescued=0      ignored=0
sles15ppc64     : ok=2   changed=1   unreachable=0   failed=0   skipped=0
rescued=0      ignored=0
ubuntu20ppc64   : ok=2   changed=0   unreachable=0   failed=0   skipped=0
rescued=0      ignored=0
ubuntu22ppc64   : ok=2   changed=1   unreachable=0   failed=0   skipped=0
rescued=0      ignored=0
ubuntu20ppc64   : ok=2   changed=0   unreachable=0   failed=0   skipped=0
rescued=0      ignored=0
ubuntu20ppc64   : ok=2   changed=0   unreachable=0   failed=0   skipped=0
rescued=0      ignored=0

```

The output from the command as seen in Example 1-8 on page 35 shows that only the target hosts `rhel8ppc64`, `sles15ppc64` and `ubuntu22ppc64` have a status value of ‘changed’, and therefore were the only hosts that required the `tcpdump` package to be installed.

We can further simplify our command-line by putting options in an `ansible.cfg` file, rather than specifying them each time we run a playbook.

An example `ansible.cfg` file you may use when running a playbook as a non-root user, and prompting for user and sudo password is shown in Example 1-9.

Example 1-9 Sample ansible.cfg file for playbook using become for privilege escalation

```

[defaults]
inventory = inventory
remote_user = ansible
ask_pass = true

[privilege_escalation]
become = true
become_method = sudo
become_user = root
become_ask_pass = true

```

An example `ansible.cfg` file that you may use when running a playbook to use SSH keys to login into target hosts directly as root is shown in Example 1-10.

Example 1-10 Sample ansible.cfg file for password-less logins as the root user

```

[defaults]
inventory = inventory
remote_user = root
ask_pass = false

```

Updating packages

A common use for Ansible playbooks is to ensure compliance by keeping packages updated with the latest security patches.

Generic package module

The generic *ansible.builtin.package* module which we used in the simple playbook shown in Example 1-5 on page 35 can also be used to update all packages as shown in the task in Example 1-11.

Example 1-11 Using generic package module to update all packages to latest

```
- name: Update all packages to latest available
  ansible.builtin.package:
    name: '*'
    state: latest
```

The generic package module may be fine for simple playbooks, but there are times where you require more fine grained control of the package updates. The next section introduces the *ansible.builtin.dnf* module which provides additional capabilities.

Using dnf module on Red Hat Enterprise Linux

There is an *ansible.builtin.yum* module, however it is written for Python 2. For newer deployments, you should use *ansible.builtin.dnf* instead. The *dnf* module allows much more control of the installation, upgrade, and removal of packages than the generic package module.

For example, you may wish to upgrade a list of packages, but only if they are already installed. The task code to do this would look something like Example 1-12.

Example 1-12 Upgrade a list of package, but do not install if not present

```
- name: Update required packages
  dnf:
    name:
      - firefox
      - curl
      - python3
    update_only: yes
    state: latest
```

Full documentation for the *dnf* module can be found in the [dnf collection description](#).

Using zypper module on SUSE Linux Enterprise Server

For those running SUSE, there is a *zypper* module that is similar to the *dnf* module for RHEL. The *zypper* module for SLES machines is part of the *community.general* collection, and needs to be installed before use. Use the **ansible-galaxy** command to install as shown in Example 1-13.

Example 1-13 Install community.general collection using ansible-galaxy

```
ansible-galaxy collection install community.general
```

Once installed, the *zypper* module options are similar to the *dnf* module. Full documentation for the *zypper* module can be found [here](#).

Using apt module on Ubuntu

The apt module options are quite similar to the dnf module. One unique option for the apt module is the `cache_valid_time` which when set to a value in seconds, will prevent the updating of repository caches until the age of the cache exceeds the value of `cache_valid_time`.

The full documentation for the `apt` module can be found [here](#).

Cross-platform playbook considerations

If you wish to write a playbook that will run on Linux hosts with different architectures, you need to be aware that some packages will not be present across all platforms.

For example, the Ubuntu packages `intel-microcode` and `amd64-microcode` are not present on ppc64 platforms. If you wanted to explicitly update these packages to the latest version in your playbook, you could use the tasks in Example 1-14 to update the packages on x86_64 platforms, but not generate an error on ppc64 platforms.

Example 1-14 Only update packages if found in 'ansible_facts'

```
- name: Update intel-microcode if present (ie not on ppc64)
  package:
    name: intel-microcode
    state: latest
    when: "'intel-microcode' in ansible_facts.packages"

- name: Update amd64-microcode if present (ie not on ppc64)
  package:
    name: amd64-microcode
    state: latest
    when: "'amd64-microcode' in ansible_facts.packages"
```

Conversely, packages from the Linux on IBM Power repository will not be present on other architectures. For more information see [IBM Linux on Power Tools](#).

1.5.2 Ansible for AIX

The supported AIX versions on IBM Power Systems are:

- ▶ AIX 7.1
- ▶ AIX 7.2
- ▶ AIX 7.3

Ansible accesses AIX hosts in the inventory via SSH, with authentication via either password, or public and private key pairs.

All nodes need to be enabled to run Open Source packages and have Python 3 installed. Beginning with AIX 7.3, Python 3 is automatically preinstalled with the operating system. On the controlling node, also Ansible version 2.9 or higher must be installed.

All of the packages can be downloaded from the [AIX Toolbox for Linux Applications](#). It is recommended to use the [DNF package](#).

This script downloads the `rpm.rte`, `dnf_bundle.tar` and `rpm.rte-4.13.0.x` which is a prerequisite for dnf. The `dnf_bundle.tar` file contains dnf and its dependent packages. This script checks if any of the packages from `dnf_bundle` are already installed and then installs the packages accordingly.

No special modules or collections are required to access an AIX host. For example, to access an AIX host named aix7.3, using a username and password, it is as simple as shown in Example 1-15.

Example 1-15 Access an AIX host using a password

```
$ ansible -m ping aix7.3 --ask-pass --user ansible
SSH password:
aix7.3 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Alternatively, to setup password-less authentication using SSH public and private keys, you can use the 'ssh-copy-id' command, as shown in Example 1-2 on page 32, to copy a preexisting public key to a AIX host.

Once the key exchange is done, you can include the private key in your node controller inventory file and execute the command directly without the need to use a password, like shown in Example 1-16.

Example 1-16 Execute command without password

```
$ ansible aix -b -m shell -a "bootinfo -s hdisk0"
aix7.3 | CHANGED => | rc=0 >>
51200
```

Ansible community for IBM AIX

There are several options for obtaining Ansible content for IBM AIX.

► Galaxy

The [Ansible Galaxy URL](#) Ansible Content for IBM Power Systems - AIX provides a collection of content used to manage and deploy Power Systems AIX.

► GitHub Repository

The [GitHub repository](#) provides modules that can be used to manage configurations and deployments of Power AIX systems. The collection content helps to include workloads on Power platforms as part of an enterprise automation strategy through the Ansible ecosystem.

► Documentation

The dedicated [documentation portal](#) offers detailed insights into the usage, configuration, and best practices of Ansible for AIX. This resource guides users through the process of integrating Ansible into their AIX workflows.

► Automation Hub

The [Automation Hub](#) offers Red Hat Ansible Certified Content for IBM Power Systems that helps you manage workloads on Power platform as part of your wider enterprise automation strategy through the Red Hat Ansible Automation Platform ecosystem. The collection is named [ibm.power_aix](#).

IBM Power Systems AIX collection

The IBM Power Systems AIX collection provides modules that can be used to manage configurations and deployments of Power AIX systems. The collection content helps to include workloads on Power platforms as part of an enterprise automation strategy through the Ansible ecosystem.

AIX specific Ansible modules

There are many AIX specific modules found at [Ansible Content for IBM Power - AIX](#) on the Ansible site some of which are shown in Table 1-7. These modules offer functions specifically targeted for AIX environments.

Table 1-7 Specific Ansible modules for AIX

Module	Minimum Ansible version	Description
_nim_upgradeios	2.9	Use NIM to update a single or a pair of Virtual I/O Servers.
aixpert	2.9	System security settings management.
alt_disk	2.9	Alternate rootvg disk management.
backup	2.9	Data or system volume group backup management.
bootlist	2.9	Alters the list of boot devices available to the system.
bosboot	2.9	Creates boot image.
chsec	2.9	Modify AIX stanza files
devices	2.9	Devices management.
emgr	2.9	System interim fixes management.
filesystem	2.9	Local and NFS filesystems management.
flrtvc	2.9	Generate FLRTVC report, download and install security and HIPER fixes.
geninstall	2.9	Generic installer for various packaging formats.
group	2.9	Manage presence, attributes and member of AIX groups.
inittab	2.9	Manage inittab entries on AIX.
installp	2.9	Installs and updates software.
internal.nim_select_target_disk	2.9	Verify/autoselect a disk used for alternate disk migration role.
lpar_facts	2.9	Reports logical partition (LPAR) related information as facts.
lpp_facts	2.9	Returns installed software products or fixes as facts.
lvg	2.9	Configure AIX LVM volume groups
lvm_facts	2.9	Reports LVM information as facts.
lvol	2.9	Configure AIX LVM logical volumes
mkfilt	2.9	Activates or deactivates the filter rules.
mktcpip	2.9	Sets the required values for starting TCP/IP on a host.

Module	Minimum Ansible version	Description
mktun	2.9	Creates, activates, deactivates and removes tunnels.
mount	2.9	Mounts/unmounts a filesystem or device on AIX.
mpio	2.9	Returns information about MultiPath I/O capable devices.
nim	2.9	Performs NIM operations - server setup, install packages, update SP or TL.
nim_backup	2.9	Uses NIM to manage backup of LPAR or VIOS clients.
nim_flrtvc	2.9	Use NIM to generate FLRTVC report, download and install security and HIPER fixes.
nim_resource	2.9	Show/define/delete NIM resource object(s).
nim_suma	2.9	Use NIM to download fixes, SP or TL from IBM Fix Central website.
nim_updateios	2.9	Use NIM to update a single or a pair of Virtual I/O Servers.
nim_vios_alt_disk	2.9	Uses NIM to create/cleanup an alternate rootvg disk of VIOS clients.
nim_vios_hc	2.9	Check if a pair of VIOSes can be updated.
nim_viosupgrade	2.9	Use NIM to upgrade VIOS(es) with the viosupgrade tool from a NIM master.
reboot	2.9	Reboot AIX machines.
smtctl	2.9	Enable and Disable Simultaneous MultiThreading Mode
suma	2.9	Download/Install fixes, SP or TL from IBM Fix Central website.
tunables	2.9	Modify/Reset/Show tunables for various components on AIX.
tunfile_mgmt	2.9	Save/Restore/Validate/Modify tunables configuration file for various components on AIX.
user	2.9	Create new users or change/remove attributes of users on AIX.

Specific Ansible roles for AIX

Table 1-8 on page 42 shows the Ansible roles designed to run specific tasks used to configure an AIX host. These tasks include activities such as configuring TCP/IP services on the AIX host, or migrating to a higher AIX level through the use of an alternate disk.

Table 1-8 Specific Ansible roles for AIX

Ansible role	Minimum Ansible version	Description
bootptab	2.9	Ansible role for modifying bootptab file on AIX.
inetd	2.9	Ansible role for enabling/disabling inetd services on AIX.
nim_alt_disk_migration	2.9	Ansible role for migrating an alternate disk to a higher AIX level.
power_aix_bootstrap	2.9	Ansible role for bootstrapping IBM Power Systems AIX servers with DNF package managers.
power_aix_vioshc	2.9	Ansible role for installing vioshc on a NIM master.

Ansible extensions for AIX

There is also an available extension for AIX that covers a reboot function which is shown in Table 1-9.

Table 1-9 Ansible plugins for AIX

Ansible Plugin	Minimum Ansible version	Description
reboot	2.9	Reboots your Logical Partition (LPAR)

1.5.3 Ansible for IBM i

IBM i remains at the forefront of innovation, serving diverse global industries with its robust capabilities. In the era of digital transformation, where swift responses are essential, IBM i users are increasingly adopting automation for tasks and cloud workload migration. This transition extends to traditional IBM i setups, Managed IT Service Providers (MSPs), and Independent Software Vendors (ISVs) such as ARCAD. To meet evolving demands for enhanced system and application management, Ansible for IBM i emerges as a significant solution.

Ansible for IBM i bridges existing gaps by providing a versatile solution that aligns with modern requirements. It equips businesses with tools to efficiently automate tasks and manage cloud workload migration. As organizations seek to extract optimum value from their IBM i environments, Ansible offers a practical approach, fostering innovation and adaptability in a dynamic digital landscape.

Why Ansible for IBM i

IBM i has been actively engaged in open-source initiatives since 2008, establishing its presence in domains such as Internet of Things (IoT) and artificial intelligence (AI). This trajectory is propelled by several influential factors. Open source acts as a channel for IBM i to explore domains such as IoT and AI, with demand surging as a new generation of engineers joins companies anchored in the IBM i ecosystem. These adept developers are positioned to redefine business applications for the IBM i platform.

The acquisition of Red Hat by IBM has sparked an interest in Ansible as a tool for automating IBM i processes and tasks. As the adoption of Ansible gains momentum, organizations are seeking IT professionals with expertise in both open-source practices and Ansible's capabilities. This intersection creates opportunities for the evolution of the IBM i platform, unlocking avenues for innovation and adaptability.

Ansible for IBM i release history

Ansible for IBM i is on a journey of continuous enhancement and evolution, aligning its capabilities with the ever-evolving landscape of IT requirements and challenges. This roadmap outlines the progressive features and functionalities introduced in different phases, showcasing how Ansible is becoming a more integral part of the IBM i ecosystem.

June 2020

In this initial phase, Ansible for IBM i set its foundation by focusing on core operational aspects:

- ▶ **PTF and LPP Management:** Ansible brought the ability to manage Program Temporary Fixes (PTFs) and Licensed Program Products (LPPs) efficiently through automation.
- ▶ **Open-Source Package Management:** The roadmap introduced support for managing open-source packages, streamlining installation and updates.
- ▶ **Object Management:** Automation was extended to manage objects on the IBM i platform, promoting a more streamlined and consistent approach.
- ▶ **PASE Support:** Ansible embraced the Portable Application Solutions Environment (PASE), enabling more versatile scripting and automation.
- ▶ **Work Management Runtime:** Ansible started facilitating work management runtime operations, enhancing control over system resources.
- ▶ **Device Management:** Automation was expanded to include device management, ensuring smoother handling of hardware resources.
- ▶ **IASP Support:** The introduction of support for Independent Auxiliary Storage Pools (IASPs) contributed to enhanced data storage capabilities.

September 2020

Building on the foundational elements, Ansible for IBM i continued to evolve with a focus on broader capabilities:

- ▶ **Advanced Fix Management:** Ansible extended its fix management capabilities, allowing more advanced and targeted fixes.
- ▶ **Basic network configuration:** Automation now covered basic network configuration tasks, ensuring network settings were managed effectively.
- ▶ **Work management:** Work management capabilities were further refined, enhancing resource allocation and workload management.
- ▶ **Security management:** Ansible incorporated security management features, contributing to robust security practices across the platform.
- ▶ **Message handling:** Automation was introduced to handle message handling tasks, ensuring timely and efficient communication.

2021

As Ansible for IBM i matured, it began addressing more strategic aspects:

- ▶ **Solution and product configuration:** The focus shifted towards solution and product configuration, enabling more comprehensive setup and customization.

- ▶ SQL services bundles: Ansible introduced bundles for SQL services, streamlining database-related operations.
- ▶ System health bundles: The roadmap incorporated system health bundles, contributing to proactive system monitoring and maintenance.

2022

As the landscape continued to evolve, Ansible for IBM i looked ahead:

- ▶ Manage in Hybrid Cloud: Ansible aimed to offer seamless management capabilities in hybrid cloud environments, ensuring consistency across on-premises and cloud deployments.
- ▶ Application Management in Cloud: The focus expanded to include application management in cloud environments, enabling efficient deployment and maintenance.
- ▶ Enhance Existing Functions: Ansible continued to enhance its existing functions, refining automation processes and expanding its capabilities.

Upcoming releases

While specific details about this release are not provided, it is indicated that more use cases and functionalities will be integrated, further enriching Ansible's offerings for IBM i users.

Ansible community for IBM i

The Ansible community for IBM i is a vibrant and active ecosystem that provides a wealth of resources to enhance automation and management capabilities for IBM i. This community-driven effort encompasses a diverse range of modules, roles, and documentation to facilitate integration of Ansible with IBM i environments.

Ansible content for IBM i:

- ▶ 50+ modules available: The Ansible community for IBM i offers a rich repository of over 50 modules, each designed to address specific tasks and functionalities.
- ▶ 10+ reusable roles: Roles, which are essentially reusable playbooks, play a pivotal role in the Ansible ecosystem. The IBM i community has contributed over 10 roles that encapsulate best practices and standard procedures for commonly performed tasks.
- ▶ Rich resources: The Ansible content for IBM i is supported by a comprehensive set of resources, ensuring users have easy access to information and guidance for effective implementation:
 - Galaxy: The [Ansible Galaxy URL](#) serves as a centralized repository for IBM i-specific content. Here, users can discover, explore, and access a wide array of modules and roles tailored for the IBM i platform.
 - GitHub Repository: The GitHub repository (<https://github.com/IBM/ibmi-oss-examples>) hosts a collection of open-source examples, showcasing practical applications of Ansible automation in IBM i environments.
 - Documentation: The dedicated documentation portal (<https://ibm.github.io/ansible-for-i/index.html>) offers detailed insights into the usage, configuration, and best practices of Ansible for IBM i. This resource guides users through the process of integrating Ansible into their IBM i workflows.
 - Automation Hub: The Automation Hub (<https://access.redhat.com/support/articles/ansible-automation-platform-certified-content>) provided by Red Hat serves as a platform for discovering and sharing Ansible automation content. It offers experience for users to explore, deploy, and manage Ansible content specific to IBM i.

Red Hat Ansible automation on IBM i

Red Hat Ansible Automation on IBM i represents a powerful synergy between two industry leaders, enabling organizations to efficiently manage IBM i workloads within the broader scope of enterprise automation. Red Hat Ansible, a renowned automation platform, offers certified content for IBM Power Systems, providing a robust solution for orchestrating and automating tasks on the IBM i platform. This certified content, available through a subscription model, aligns IBM i workloads with the Red Hat Ansible Automation Platform ecosystem.

Certified integration between Ansible and IBM Power Systems:

- ▶ **Unified automation strategy:** The integration of Red Hat Ansible with IBM Power Systems offers a certified pathway for holistic automation across a diverse infrastructure landscape. This integration encompasses multiple platforms, including AIX, IBM i, and Linux on Power, enabling enterprises to unify their automation strategies under the Red Hat Ansible Automation Platform.
- ▶ **Certified content repository:** The Red Hat Ansible Automation Platform provides a certified repository of content tailored for IBM Power Systems. This repository equips organizations with ready-to-use automation playbooks, modules, and roles designed to efficiently manage and orchestrate IBM i workloads.
- ▶ **Solution benefits:**
 - **Consistency:** The certified integration ensures uniform automation practices across heterogeneous environments. With Red Hat Ansible Automation, organizations can establish consistent automation workflows that span IBM Power Systems, AIX, Linux on Power, and beyond.
 - **Transparency:** The unified approach offered by Red Hat Ansible fosters transparency in automation operations. Organizations gain a comprehensive view of their IBM i workloads alongside other systems, promoting a clearer understanding of automation tasks and results.
 - **Skills enhancement:** By embracing Red Hat Ansible Automation for IBM i, IT teams can enhance their skill sets and proficiency in modern automation practices. The platform offers a standardized and adaptable automation framework that empowers teams to efficiently manage complex workloads.

Note: The integration of Red Hat Ansible Automation with IBM Power Systems reflects a commitment to the automation processes, improving operational agility, and enabling enterprises to confidently manage their IBM i workloads. This collaboration provides a powerful tool set for organizations seeking to navigate the complexities of enterprise automation with consistency, transparency, and advanced skills development. For more details and insights into this integration, further information can be accessed through the provided links:

- ▶ *Red Hat Ansible Integration with IBM Power Systems*
- ▶ *Red Hat Ansible Automation Hub for IBM Power Systems.*

IBM i - specific Ansible modules

Within Ansible version 2.9 and beyond, a dedicated set of modules tailored specifically for IBM i systems is available, as depicted in Table 1-10 on page 46. While these modules constitute a smaller subset compared to the complete range present in Ansible, they offer targeted functionality designed to cater to the unique requirements of IBM i environments

Table 1-10 Specific Ansible modules for IBM i

Module	Minimum Ansible version	Description
ibmi_at	2.9	Schedule a batch job on a remote IBM i node.
ibmi_cl_command	2.9	Runs a CL command.
ibmi_copy	2.9	Copies a save file from local to a remote IBM i node.
ibmi_display_subsystem	2.9	Displays all active subsystems or active jobs in a subsystem.
ibmi_end_subsystem	2.9	Ends a subsystem.
ibmi_start_subsystem	2.9	Starts a subsystem.
ibmi_lib_restore	2.9	Restores one library on a remote IBM i node.
ibmi_lib_save	2.9	Saves one library on a remote IBM i node.
ibmi_reboot	2.9	Restarts the IBM i machine.
ibmi_sql_execute	2.9	Runs an SQL non-DQL (Data Query Language) statement.
ibmi_sql_query	2.9	Runs an SQL DQL (Data Query Language) statement.
ibmi_fix	2.9	Loads from save file, and applies, removes, or queries PTFs.
ibmi_fix_imgclg	2.9	Installs fixes from a virtual image.
ibmi_object_find	2.9	Finds a specific IBM i object.
ibmi_submit_job	2.9	Submits an IBM i job.
ibmi_iasp	2.9	Controls an independent auxiliary storage pool (IASP) on a target IBM i node.
ibmi_tcp_interface	2.9	Manages the IBM i TCP interface. You can add, remove, start, end, or query a TCP interface.
ibmi_tcp_server_service	2.9	Manages a TCP server on a remote IBM i node.

Shared core modules with IBM i support

These modules, termed “common modules” are aptly named due to their compatibility with a range of operating systems. These modules are harnessed by IBM i as well. Notably, core modules extend their support to IBM i via PASE (Portable Application Solutions Environment). Table 1-11 on page 47 shared core modules with IBM i compatibility.

Table 1-11 Shared core common modules supported on IBM i

Common modules	Minimum Ansible version	Description
assemble	2.9	Assembles content from different sources into a single file or variable
authorize_key	2.9	Adds or removes SSH authorized keys for specified users
blockinfile	2.9	Inserts or updates a block of text surrounded by customizable markers
command	2.9	Executes shell commands on target hosts
copy	2.9	Copies files to remote locations
fetch	2.9	Fetches files from remote locations
file	2.9	Manages files and file properties on remote hosts
find	2.9	Searches for files in a directory hierarchy
git	2.9	Manages git repositories on remote hosts
lineinfile	2.9	Ensures a particular line is in a file, or replaces an existing line
pause	2.9	Pauses a playbook for a specified amount of time
ping	2.9	A basic connectivity test to target hosts
pip	2.9	Manages Python packages using pip
script	2.9	Runs a local script on the remote hosts
setup	2.9	Gathers system information from target hosts
shell	2.9	Executes shell commands on target hosts
stats	2.9	Gathers facts about remote hosts
synchronize	2.9	Synchronizes files/directories to remote hosts
wait_for_connection	2.9	Waits for a host to become reachable

Specific Ansible roles for IBM i

Ansible Role encompasses a collection of tasks designed to configure an IBM i host for various common tasks. These tasks include activities such as applying PTFs (Program Temporary Fixes) and other configuration procedures on the IBM i platform. Roles are articulated using YAML files within a structured directory layout. This layout typically comprises sections such as defaults, vars, tasks, files, templates, and more. In Table 1-12 you find a selection of Ansible roles specifically tailored for IBM i:

Table 1-12 *Specific Ansible roles for IBM i*

Ansible role	Minimum Ansible version	Description
apply_all_loaded_ptfs	2.9	Apply all loaded Program Temporary Fixes (PTFs) on IBM i hosts
apply_ptf	2.9	Apply specific Program Temporary Fixes (PTFs) on IBM i hosts
change_server_state_via_powervc	2.9	Change server state using PowerVC on IBM i hosts
configure_passwordless_ssh_login	2.9	Configure passwordless SSH login on IBM i hosts
deploy_vm_via_powervc	2.9	Deploy IBM i virtual machines in PowerVC
display_network_info_via_powervc	2.9	Display network information via PowerVC on IBM i hosts
display_vm_info_via_powervc	2.9	Display virtual machine information via PowerVC on IBM i hosts
download_individual_ptfs	2.9	Download individual Program Temporary Fixes (PTFs) on IBM i
present_ip_interface	2.9	Present IP interface configuration on IBM i hosts

Note: To explore additional Ansible roles tailored for IBM i, you can find a *comprehensive collection*. This repository hosts a variety of roles designed to facilitate IBM i-specific tasks, providing a valuable resource for enhancing your automation capabilities on the platform.

Ansible extensions for IBM i

Ansible extensions for IBM i comprise code components that complement the core functionalities of Ansible, enriching its capabilities. These extensions, often referred to as plugins, serve as dynamic tools to enhance flexibility and expand the feature set of Ansible.

Note: Finding these plugins is straightforward. You can explore them through the following links:

- ▶ [Ansible Galaxy - IBM Power i Plugins.](#)
- ▶ [GitHub repository for Ansible IBM i Plugins.](#)

In Table 1-13, you find a selection of Ansible plugins specifically designed for IBM i. These plugins cover various functionalities, such as copying files, interacting with IBM DB2® on IBM i, utility functions, fetching data, and rebooting operations.

Table 1-13 Ansible plugins for IBM i

Ansible Plugin	Minimum Ansible Version	Description
ibmi_copy	2.9	Copies files and directories on the IBM i system.
ibmi_db2i_tools	2.9	Provides tools for managing DB2 on IBM i.
ibmi_ibmi_module	2.9	Offers various IBM i specific modules for Ansible tasks.
ibmi_ibmi_util	2.9	Includes utility functions for IBM i tasks.
ibmi_fetch	2.9	Fetches files and URLs on the IBM i system.
ibmi_reboot	2.9	Initiates system reboots on the IBM i platform.

Ansible playbooks for IBM i

An Ansible playbook for IBM i serves as a structured set of automation tasks designed to run with minimal to no human intervention. These playbooks, crafted in YAML format, consist of mappings and sequences, and they incorporate Ansible modules to execute specific actions. A prime feature of Ansible playbooks is their ability to automate intricate workflows on the IBM i platform effortlessly.

Table 1-14 illustrates a selection of Ansible playbooks tailored for IBM i:

Table 1-14 Ansible playbooks for IBM i

Ansible playbooks	Minimum Ansible version	Description
enable-ansible-for-i.yml	2.9	Configures prerequisites on IBM i endpoints for utilizing Ansible for IBM i collections.
enable_offline_ibmi.yml	2.9	Enables Ansible automation for IBM i endpoints without requiring internet access.
ibmi-sql-sample.yml	2.9	Demonstrates usage of Ansible for IBM i to execute SQL statements on IBM i systems.
ibmi-sysval-sample.yml	2.9	Illustrates Ansible for IBM i to query system values on IBM i platforms.
ssh-addkey.yml	2.9	Facilitates the addition of SSH keys on IBM i systems, enabling secure communication for Ansible operations.

Note: To explore practical examples, IBM i users can refer to the samples available [here](#)

Ansible inventory on IBM i

An Ansible inventory serves as a configuration file that defines individual hosts and groups of hosts within your environment. This allows you to manage multiple systems within your infrastructure simultaneously. For instance, you can create inventory groups that reflect different parts of your infrastructure, making it easier to target specific hosts or sets of hosts.

In the context of IBM i, your inventory file can define various groups such as “IBM Power Virtualization Center” and “IBM i systems”. This provides a clear structure for managing and orchestrating tasks across different systems. Each group is associated with specific attributes, including connection details and authentication credentials.

Example 1-17 shows an inventory setup where you have two groups: “powervc_servers” and “IBM i”. The “powervc_servers” group includes details about the Power Virtualization Center servers, specifying the SSH host, username, password, and Python interpreter. The “IBM i” group, on the other hand, outlines the connection information for your IBM i system.

Example 1-17 Sample Ansible inventory configuration for IBM i and PowerVC

```
[powervc_servers]
powervc ansible_ssh_host=your_powervc_ip ansible_ssh_user=your_user
ansible_ssh_pass=your_password
ansible_python_interpreter="python3"

[ibmi]
source ansible_ssh_host=your_source_ibmi_ip
ansible_ssh_user=your_user ansible_ssh_pass=your_password
```

By using this approach, you can efficiently manage and automate tasks across a diverse set of systems within your infrastructure. This ensures that Ansible can interact with the specified hosts, the process of configuration management and orchestration.

1.5.4 Ansible for IBM Power Hardware Management Console

The IBM Power Hardware Management Console (HMC) is rapidly evolving, with major advances being made across Power Systems. In today’s Information Technology world, automation is of paramount importance to save time and increase efficiencies. The power-hmc collection is part of IBM’s continuous efforts to adapt HMC to the ever-evolving automation trends in the IT infrastructure administration landscape.

Power HMC Ansible content helps administrators include Power HMC as part of their automation strategies through the Ansible ecosystem. Using Power HMC Ansible content in IT automation helps maintain a consistent and convenient management interface for multiple Power HMCs and Power servers.

Power HMC Ansible content ‘modules can be leveraged to do Power HMC patch management, Logical Partition management, Power Server management, password policy configurations and HMC-based Power systems dynamic inventory building.

HMC specific Ansible modules

Table 1-15 provides a summary of modules currently supported.

Table 1-15 Specific Ansible modules for the IBM Power Hardware Management Console (HMC)

Module	Minimum Ansible version	Description
hmc_command	2.9	This module can be used to run HMC commands
hmc_pwdpolicy	2.9	The password policy module is to manage the Power HMC password policy.
hmc_update_upgrade	2.9	The HMC patch management module can be used to perform update or upgrade of HMC.
hmc_user	2.9	Manage the users on the HMC
power_system	2.9	The Power System management module can be used to power cycle the system, modify configurations, and modify resources.
powervm_dlpar	2.9	This module can be used to dynamically configure Processor, Memory, and Storage settings.
powervm_inventory	2.9	This is a dynamic inventory plugin which helps in dynamically building inventory of Power Systems and partitions connected to HMC.
powervm_lpar_instance	2.9	The logical partition management module helps with the creation, deletion, activate, and shutdown of logical partitions.
powervm_lpar_migration	2.9	The logical partition migration module can be used to validate, and migrate logical partitions.
vios	2.9	The Virtual IO Server module helps to create and install Virtual I/O Server.

Ansible extensions for the HMC

There are also extensions available for the HMC which are shown in Table 1-16.

Table 1-16 Ansible plugins for the HMC

Ansible plugin	Minimum Ansible version	Description
hmc_resource	2.9	remove try back from list_all_managed_system_details method
hmc_exceptions	2.9	Change custom exception name
hmc_rest_client	2.9	Add unit tests and document correction
hmc_cli_client	2.9	Fix password special character issue
hmc_command_stack	2.9	Fix for show ldap details in configure ldap action
powervm_inventory	2.9	HMC-based inventory source for Power servers

The ansible-power-hmc collection requires that you be running HMC V10, HMC V9R1 or later, or HMC V8R8.7.0 or later. It supports Ansible 2.9 or later and requires Python 3.

More information about the collection can be found in this [BLOG](#).

1.5.5 Ansible for Power Virtual I/O server

IBM PowerVM Virtual I/O Server (VIOS) is a special IBM AIX based appliance for IBM Power systems which helps to virtualize storage and network adapters for virtual machines running on the managed system. The VIOS should be installed in pairs to provide the highly available enterprise system expected from your Power environment. In addition, you may have business requirements for separation of applications that requires additional VIOS pairs for security. This means that you have at least two VIOS partitions for each physical server that you are managing.

This is a perfect case for Ansible management as you need to ensure that all of your VIOS LPARs are maintained and updated as required as new updates come out. In addition, if you are building a new bare metal environment, it is helpful to be able to ensure that the VIOS image used in the new server is the level that you want. Starting with VIOS 4.1, VIOS will be enabled for Ansible out of the box, making your Ansible setup significantly easier.

The power-vios collection provides the functions you need to manage your VIOS environment such as install a new VIOS, manage the security parameters on your VIOS, backup and restore VIOS images, and upgrade VIOS code.

VIOS specific Ansible modules

Table 1-17 provides a summary of modules currently supported.

Table 1-17 Specific Ansible modules for VIOS

Module	Minimum Ansible version	Description
mapping_facts	2.9	Returns the mapping between physical, logical, and virtual devices as facts
updateios	2.9	Updates the Virtual I/O Server to the latest maintenance level
viosbr	2.9	Backup/Restore the configuration of the VIOS
backupios	2.9	Creates an installable image of the root volume group
viosupgrade	2.9	Upgrades the Virtual I/O Server
alt_root_vg	2.9	Create/Cleanup an alternate rootvg disk on a VIOS
viosecure	2.9	Configures security hardening rules and firewall

Ansible extensions for the VIOS

There is also an extension available for the VIOS which is shown in Table 1-18.

Table 1-18 Ansible plugins for the VIOS

Ansible plugin	Minimum Ansible version	Description
viosupgrade	2.9	Update viosupgrade.py

1.5.6 Ansible for IBM Power Virtual Server

IBM Power Systems Virtual Server on IBM Cloud offers a rapid and efficient means to create and deploy virtual machines (VMs) across various operating systems, including AIX, IBM i, and Linux tailored for Power Systems. This solution stands out for its robust security measures and the ability to scale compute capacity as needed.

Enterprise Infrastructure-as-a-Service offering

This service utilize POWER resources distributed globally, ensuring low-latency connectivity to IBM Cloud infrastructure. It features dedicated components, including a distinct network and direct attached storage, enhancing reliability and performance.

Collaborative Cloud offering

At its core, this offering represents an Infrastructure-as-a-Service (IaaS) approach, reflecting the evolving IT landscape where cloud capabilities are increasingly essential for resource consumption. IBM recognizes its extensive client base relying on AIX and IBM i over the years, making this solution a vital addition.

Architectural overview

Within the IBM Cloud environment, the IBM Power Virtual Server resides as a distinct entity. Picture it as a specialized collocation site within one of our IBM SoftLayer® or cloud data centers. Here, you find a segregated enclosure housing all the POWER equipment. A significant advantage of IBM Power Virtual Servers is the consistency of their architecture with on-premises Power Systems. This means that if you are configuring Power Systems on-premises today, you can rest assured that it is a similar process to set up an IBM Power Systems Virtual Server. To illustrate the architecture refer to Figure 1-12.

For example, redundant Virtual I/O servers and NPIV attached storage, as seen in on-premises setups, are mirrored in Power virtual server environments. Redundant SAN fabric, Hardware Management Console (HMC), and PowerVC configurations are all consistent between on-premises and cloud deployments.

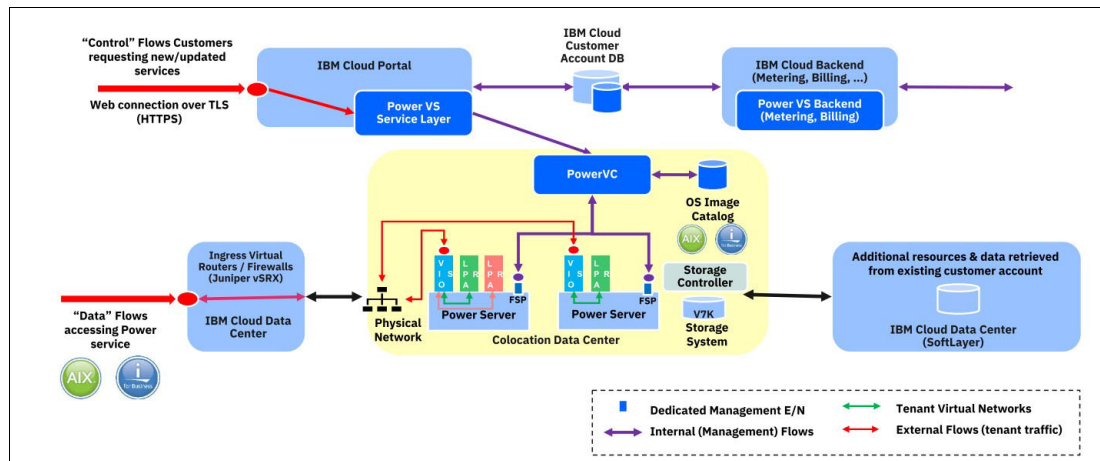


Figure 1-12 Architecture of service

Infrastructure as a Service

Transitioning Power resources to a cloud data center is not as straightforward as a simple relocation. This is where the concept of a collocation (COLO) site comes into play. The COLO serves as the foundation for Infrastructure-as-a-Service (IaaS). In this context, IaaS encompasses everything beneath the virtual machine layer. This includes the PowerVM

hypervisor, firmware, Virtual I/O server, HMC, PowerVC, network switches, and SAN switches, all of which are integral parts of the IaaS.

So, how do you manage this environment if you cannot directly access PowerVC or the SAN switches? *You interface with a service layer that implements the open service broker framework.* This framework is a standard across various cloud portals, such as GCP and Azure. Essentially, it is a means of managing services in the cloud.

This service layer offers multiple interfaces, including a command line and a REST API. While the interface may change, you retain the same core capabilities you are accustomed to on-premises. For instance, if you have scripts that interact with the HMC today, you will need to adapt them to the IBM Cloud command line. However, the functionality you rely on remains available.

Ansible integration with IBM PowerVS

Within the PowerVS architecture, PowerVC plays a central role. It enables integration with various DevOps tools like Terraform, Ansible, Puppet, among others, and offers an API for enhanced automation capabilities.

Specifically, when considering Ansible, it permits us to provision AIX and IBM i instances within IBM PowerVS. Here is how to go about it.

IBM Power Virtual Server in IBM Cloud

In this example, we demonstrate the creation of a Power Systems Virtual Server running AIX or IBM i. This server is configured to allow incoming SSH connections through a publicly accessible IP address, authenticated using the provided SSH key.

Power Systems Virtual Server resources

The following infrastructure resources will be established using Ansible modules:

- ▶ SSH Key (`ibm_pi_key`)
- ▶ Network (`ibm_pi_network`)
- ▶ Virtual Server Instance (`ibm_pi_instance`)

Configuration parameters

Users have the flexibility to set the following parameters:

- ▶ `pi_name`: The name assigned to the Virtual Server Instance.
- ▶ `sys_type`: The type of system on which to create the VM (e.g., s922, e880, any).
- ▶ `pi_image`: The name of the VM image (users can retrieve available images).
- ▶ `proc_type`: The type of processor mode in which the VM will run (shared or dedicated).
- ▶ `processors`: The number of vCPUs to assign to the VM (as visible within the guest operating system).
- ▶ `memory`: The amount of memory (in GB) to assign to the VM.
- ▶ `pi_cloud_instance_id`: The `cloud_instance_id` for this account.
- ▶ `ssh_public_key`: The value of the SSH public key to be authorized for SSH access.

Running the playbook

Before proceeding, ensure you've set your API Key and Region:

1. Obtain an IBM Cloud API key.
2. Export your API key to the `IC_API_KEY` environment variable: **export IC_API_KEY=<YOUR_API_KEY_HERE>**

Note: While modules also support the `ibmcloud_api_key` parameter, it is recommended to use environment variables when encrypting your API key value.

3. Export your desired IBM Cloud region to the `IC_REGION` environment variable: **export**
`IC_REGION=<REGION_NAME_HERE>`

Note: Modules also support the `region` parameter.

4. Export your desired IBM Cloud zone to the `IC_ZONE` environment variable: **export**
`IC_ZONE=<ZONE_NAME_HERE>`

Note: This is used for multi-zone supported power instances.

With the environment variables configured, follow these steps:

Create resources

To create all resources and test public SSH connections to the VM, run the `create` playbook:

1.5.7 Ansible for applications

As we have already discussed, Ansible can simplify the installation and operation of your infrastructure components, including creating LPARs running AIX, IBM i, or Linux on Power. However, Ansible can also provide a significant reduction in the time and effort required to manage many complex application environments.

Using Ansible for automation in these application environments can reduce the amount of time your support staff spends on basic tasks like installing or modifying your application environment and allows you to have a build a consistent and secure environment for your application instances, while allowing your support staff to concentrate on tasks that are more important and that can drive new business opportunities for your company. There are many application environments that can be managed by Ansible. We will describe how Ansible can make your team more productive when managing two of the most common applications that run on IBM Power systems.

Ansible for Oracle

Oracle Database has been a leading enterprise database management system for over three decades. Despite the emergence of new technologies and competitors, Oracle Database remains a popular choice for many organizations.

The Oracle Database is known for its ability to manage and process large amounts of data quickly and efficiently. As a result, Oracle Database is widely used across different industries, from finance to healthcare, and is trusted by organizations of all sizes to store and manage critical business data. Oracle Database has established itself as a reliable and versatile database management system that can meet the complex data needs of modern enterprises. Its performance capabilities, scalability, security features, and integration with cloud technologies make it a top choice for organizations.

One of the common platforms for running Oracle databases and application is on IBM Power running the AIX operating system. With automation becoming the norm in IT operations, installation and administration of Oracle database on AIX is not an exception. There now exist Ansible collections for installation operations both on a single-node machine as well as on Oracle RAC in addition to a collection for automating DBA administrative operations.

Advantages of using Ansible for Oracle

The Ansible collections for Oracle on AIX provide the following benefits:

► **Automated database installation**

Even when creating a single instance database, setting up Oracle Database on AIX involves multiple manual processes, including defining the LPAR, installing AIX, configuring the filesystem (either JFS2 or ASM), configuring the network connections and generally setting up system configuration values appropriate for running your database environment.

Setting up an Oracle Real Application Clusters (RAC) on AIX involves setting up an AIX environment on the hosts that meet RAC's specific requirements from kernel tunables, network attributes, shared disk attributes, passwordless to user equivalent ssh connections etc. The manual process to accomplish these tasks is tedious and error prone. During the Grid and Database install, the GUI frequently prompts for entering input that ties up the user for a long time.

Save time with infrastructure automation. The whole installation can take two days for seasoned users. With the help of Ansible Oracle RAC ASM collection, it takes typically 5 hours to complete a 4-node RAC installation, a tremendous time saving. It is completely hands-free and can consistently recreate Oracle RAC for other projects. The value of this collection helps your organization to significantly improve productivity for installation tasks.

► **Automated database management**

After your databases are installed and operational, there are still many tasks that need to be done to manage the environments. As the number of database instances grows, the amount of time required for day to day operation also grows. The Ansible Oracle DBA package provides you the ability to automate the day to day tasks managed by your DBAs to increase their productivity and allow them to focus on more business critical issues.

The ODBA collection allows you to add/drop databases, manage users, manage space (ASM and ACFS functions), manage patches, and otherwise manage your Oracle environment. In addition, the ODBA collection can automate Oracle upgrades.

Ansible for SAP

Ansible is an open-source automation tool that is used to simplify the management and deployment of IT infrastructures. It is especially useful when it comes to managing complex systems like SAP. In today's digital world, if an organization relies on SAP HANA and SAP S/4HANA for its business-critical operations, downtime can result in not only revenue loss but also performance and service degradation, increased security exposure, and poor end user experience. This not only affects your ROI, but it also acts to distract your SAP teams from more strategic, high-priority projects.

Figure 1-13 on page 57 illustrates the breadth of function that can be provided by Ansible for automating your SAP land scape.

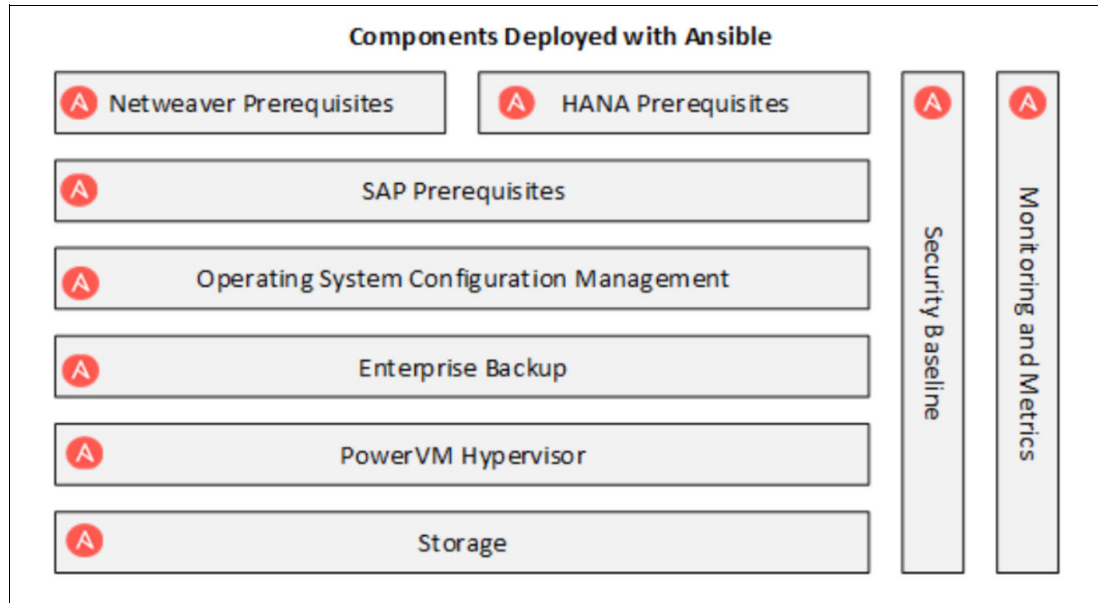


Figure 1-13 Components that can be automated using Ansible¹⁶

Red Hat Ansible Automation Platform eliminates these common obstacles, with an intuitive interface and ready-to-use, trusted content that is custom-built for SAP migrations. Ansible can also be integrated with SAP to streamline operations, improve efficiency, and reduce manual tasks. With Ansible Automation Platform, manual tasks that used to take days can be done in hours or even minutes. By consolidating on a single, unified platform, your teams can more easily share automation content and workflows, and scale as your organization evolves and uncovers new automation use cases. Firstly, let's understand what SAP is. SAP stands for Systems, Applications, and Products in Data Processing. It is a software suite that covers various business processes such as finance, logistics, human resources, and more. SAP is widely used by organizations of all sizes and industries to manage their operations effectively.

SAP installations can be quite complex and require skilled administrators to manage and maintain them. This usually involves performing various tasks such as system configuration, installation of patches, managing user accounts, monitoring system performance, and more.

Advantages of Using Ansible for SAP

The following list enumerates some of the advantages of using Ansible to manage your SAP environment.

► **Rapid Deployment and Configuration:**

Deploying and configuring SAP systems, such as SAP HANA and S/4HANA, can be complex and time-consuming. Ansible simplifies this process by automating the deployment of SAP software, along with the necessary configurations. With Ansible playbooks, you can define the desired system settings, network configurations, and more, ensuring consistency across all your SAP systems. Furthermore, Ansible allows for reusability, making it easy to replicate configurations across different environments.

► **Continuous Software Delivery:**

Ansible streamlines the software deployment process, enabling continuous delivery of updates, patches, and enhancements. With Ansible playbooks, you can automate the entire software installation, patching, and upgrade processes. Ansible's ability to perform tasks in parallel helps save time by executing these operations on multiple systems

¹⁶ <https://www.adventone.com/sap-hana-at-the-speed-of-ansible/>

simultaneously. This ensures that your SAP systems are always up to date, with minimal disruption and maximum efficiency.

► **Intelligent Monitoring and Maintenance:**

Monitoring your SAP landscape is crucial for identifying potential issues and preventing system downtime. Ansible integrates seamlessly with monitoring tools, allowing you to automate monitoring processes and trigger alerts or actions based on predefined thresholds. By proactively monitoring performance metrics, system errors, and other vital indicators, you can ensure the availability and reliability of your SAP environment.

► **Enhanced High Availability and Disaster Recovery:**

Ensuring high availability and disaster recovery capabilities for SAP systems is paramount for minimizing business interruptions. Ansible enables you to automate the configuration and management of HA and DR environments. Whether it's setting up system replication between primary and secondary nodes or automating failover and failback processes, Ansible streamlines these tasks and reduces the risk of human errors.

► **Scalability and Flexibility:**

Ansible empowers you to scale your SAP infrastructure to handle growing business demands efficiently. With Ansible playbooks, you can define rules and conditions for scaling up or down system resources, such as adding or removing compute nodes or adjusting memory allocations. This flexibility allows you to optimize resource utilization and accommodate changing workloads effortlessly.

Managing SAP systems can be complex and time-consuming, but with the power of Ansible, organizations can revolutionize their SAP operations. By automating tasks such as system configuration, software deployment, monitoring, and scaling, Ansible simplifies SAP management, enhances efficiency, and drives agility in your SAP landscape. With the Red Hat Ansible Automation Platform, you can streamline your SAP operations, reduce manual effort, ensure consistency, and improve overall productivity and reliability of your SAP environment.



Ansible architecture and design

Understanding how to set up Ansible to manage the automation in your environment is important to the ultimate success of your automation journey. Although Ansible is easy to install and start using, it is still important that you set up the Ansible environment to support your staff and meet your business needs. In this chapter, we discuss the different components of Ansible and present some hints and tips for you to consider as you build out your Ansible environment.

The following topics are covered in this chapter:

- ▶ Ansible architecture and components
- ▶ Understanding Ansible's declarative language
- ▶ Understanding Ansible Inventory
- ▶ Ansible tasks, playbooks and modules
- ▶ Ansible roles and collections
- ▶ Preferred practices for playbook and role design
- ▶ Versioning and documenting playbooks and roles
- ▶ Testing and validating playbooks and roles

2.1 Ansible architecture and components

Introduction

In early releases, Ansible was installed as a package that included core components and plugins that allowed you to control automation on different platforms. It was determined that this was too complex to maintain which introduced additional difficulties and delays in updating the Ansible package. Now, Ansible is released in multiple packages which can be installed independently. Figure 2-1 shows some of the Ansible components.

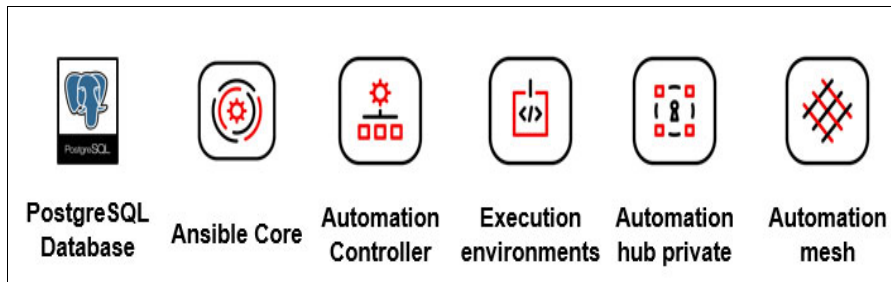


Figure 2-1 List of Ansible automation components

One option for building your environment is to use the automation base package - Ansible Core (previously called Ansible Engine or Ansible base) and use Ansible Automation Controller as the command-line interface (CLI).

Alternatively, the automation environment can be built with Red Hat Ansible Automation Platform (RHAAP) together with Ansible Core. Automation Platform has a graphical user interface (GUI) and also extends Ansible functionality with additional management capabilities. For more details refer to section 1.3.2, “Red Hat Ansible Automation Platform” on page 10.

You should consider several things before deciding how to set up your automation environment. These considerations will help you to choose the right Ansible products, as you define and build the proper architecture that will meet your business requirements. Consider the following:

- ▶ **Computing resource availability and cost**

Understand the number of systems that will be managed and the required availability of your Ansible management infrastructure. This will impact the number of Ansible systems required to manage your environment, including the required resources (CPU, Memory, Disk etc).

- ▶ **Administrative environment**

Is a command line interface (CLI) acceptable or do you need a graphical user interface (GUI) for user friendliness.

- ▶ **Security and compliance**

Understand your security and compliance requirements for user authentication and access control. Do you need to separate automation into multiple management domains to comply with security & compliance guidelines.

- ▶ **High availability and scalability**

Do you need high availability in your automaton? If you are monitoring and managing critical business functions, the answer is probably yes. Be careful to design and build the automation so that it will scale to manage additional environments or handle growth in the number of machines and tasks being run.

► **Integration and compatibilities**

Do you need to integrate with other solutions and services? is your solution compatible with the hosts and systems that are targeted for automation.

► **Complexity of Day-2 operations**

Understand how complex is it to manage the automation environment operational activities. Especially consider automation resources, like:

- Credentials to access target systems.
- Skilled resources to develop and manage playbooks.
- Take time to define inventories, hosts and hosts groups to simplify the automation environment.
- Resources to manage playbook execution and access control.

Once you have considered all of these factors, you can plan and design the automation environment based on your requirements in these areas. More detail on designing your Ansible environment, including reference architectures, is shown in section 3.1, “Architecting your Ansible environment” on page 100.

2.1.1 Controller and client functions

From an administrative view, there are two primary functions of all the systems in the automation environment. A system is either a controller or a system is a client – a system that is being managed.

Controller functions

A controller is the machine or set of machines which run the Ansible tools (ansible-playbook, Ansible, ansible-vault and others) and automation tasks. Depending on your solution, the controller will run the Ansible CLI or it can be using a graphical user interface. Using a GUI often simplifies the management of your client inventory, job templates, and workflow templates as well as simplifying how you launch jobs, schedule workflows, and track and report changes. The requirements for being an Ansible controller are discussed in section 3.2, “Choosing the Ansible controller node” on page 117.

Client Functions

The machine or set of machines, which are being managed by Ansible. They are also referred to as 'hosts' or managed nodes and are servers, network appliances or other managed devices. Ansible does not need to be installed on the managed nodes as the Ansible control nodes are used to run the automation tasks. The prerequisites for being a managed node is discussed in section 3.4, “Preparing your systems to be Ansible clients” on page 144.

Controller to client connectivity

The Ansible controller is responsible for automating task execution on the managed nodes or target devices (servers, network appliances or any computer), Ansible works by connecting the controller to the managed nodes and pushing out small programs, called “Ansible modules,” to them. These programs are written to define the desired state of the client and then allow the client to make the required changes to ensure that it meets the desired state.

The communication between Ansible controllers to managed nodes or the target devices can be different depending on the target devices. For example:

- Linux/Unix hosts use SSH by default,
- Windows hosts use WinRM over HTTP/HTTPS by default.
- Network devices use CLI or XML over SSH.
- Appliance or web-based services could use RESTful API over HTTP/HTTPS.

2.2 Understanding Ansible's declarative language

Ansible playbooks, which are used to define automation tasks, are written in YAML. YAML is used to describe the tasks, expected configuration, and data structures in a human-readable format. YAML is a recursive acronym for *YAML Ain't Markup Language*. In this section, we'll explore how Ansible utilizes YAML to define automation playbooks and provide some simple examples to help you get started with effectively building automation tasks.

2.2.1 YAML Structure

YAML's design is to create a human-readable format. YAML files consist of key-value pairs, lists, and nested structures. Indentation is important in YAML as indentation is used to define the hierarchy of commands and lists.

When you create a YAML file consider these basic rules:

- YAML is case sensitive.
- The files should have `.yaml` or `.yml` as the extension.
- Indentation is critical to define the hierarchy of the YAML code:
 - Do not use tabs for indentation, you must use spaces instead.
 - While not required, it is recommended to use two spaces per indent. Many editors can be set to provide spacing when editing YAML files.
- While not required, playbooks start with `---` to denote the beginning of a document and `...` to denote the end of a document.
- There are important key sequences to define Ansible code:
 - Comments are denoted by a space followed by hash tag (`#`). Any text following the space hash tag will be ignored.
 - A colon followed by a space (or newline) `:`: `"` is an indicator for a mapping.
 - To have any of these items in a string and not be interpreted by Ansible, use single quotes or double quotes around the full value string.
- Extra lines are ignored and can be used to improve readability of the playbook.

For Ansible, nearly every YAML file starts with a list. Each item in the list is a list of key/value pairs, commonly called a “hash” or a “dictionary”. So, we need to know how to write lists and dictionaries in YAML:

- ▶ All members of a list are lines beginning at the same indentation level starting with a `-` (a dash and a space) as shown in Example 2-1.

Example 2-1 YAML list definition

```
---
# A list of tasty fruits
- Apple
- Orange
- Strawberry
- Mango
...
```

- ▶ A dictionary is represented in a simple key: value form (the colon must be followed by a space) as shown in Example 2-2.

Example 2-2 YAML dictionary definition

```
---
# An employee record
martin:
```

```
name: Martin D'vloper
job: Developer
skill: Elite
...
```

In the playbook in Example 2-3, the lines *name* and *hosts* are simple key-value pairs. Whereas the line *tasks* is a list. A list can contain other objects which are defined with key-value pairs organized in a hierarchy.

Example 2-3 Example playbook

```
---
- name: Simple Ansible Playbook
  hosts: web_servers
  tasks:
    - name: Ensure Apache is installed
      apt:
        name: apache2
        state: present

    - name: Start Apache service
      service:
        name: apache2
        state: started
...
```

Variables

You can define variables in Ansible to make your playbooks more dynamic and reusable. Variables can be defined at different levels, including playbook, group, and host variables. In Example 2-4 the *vars* section defines a playbook-level variable *http_port*.

The template module is used to copy a configuration template (`apache.conf.j2`) to the destination and notify the `Restart Apache` handler. You might have noticed `j2` in the code example which refers to Jinja templating language that is quite common in Python development. See section 2.2.2, “Jinja2” on page 64 for additional details on Jinja templating.

Example 2-4 Playbook with variables

```
---
- name: Ansible Playbook with Variables
  hosts: web_servers
  vars:
    http_port: 80
  tasks:
    - name: Ensure Apache is installed
      apt:
        name: apache2
        state: present

    - name: Configure Apache
      template:
        src: apache.conf.j2
        dest: /etc/apache2/apache2.conf
      notify: Restart Apache
```

```

- name: Start Apache service
  service:
    name: apache2
    state: started
handlers:
- name: Restart Apache
  service:
    name: apache2
    state: restarted

```

Conditionals and Loops

Ansible supports conditionals and loops to make your playbooks more flexible. Example 2-5 shows a simple example.

In this playbook *http_ports* is a variable which is a list of ports. The *loop* keyword is used to iterate over the list and generate configuration files for each port.

Example 2-5 Playbook with variable and loop

```

---
- name: Ansible Playbook with Conditionals and Loops
  hosts: web_servers
  vars:
    http_ports:
      - 80
      - 8080
  tasks:
    - name: Ensure Apache is installed
      apt:
        name: apache2
        state: present

    - name: Configure Apache
      template:
        src: apache.conf.j2
        dest: "/etc/apache2/apache{{ item }}.conf"
        loop: "{{ http_ports }}"
        notify: Restart Apache

    - name: Start Apache service
      service:
        name: apache2
        state: started
  handlers:
    - name: Restart Apache
      service:
        name: apache2
        state: restarted

```

2.2.2 Jinja2

Jinja is a modern and widely used template engine for Python. It was developed by Armin Ronacher, the creator of the Flask web framework, and was first released in 2008. Jinja2 is used for generating dynamic content, such as HTML, XML, JSON, YAML and other text-based formats, by incorporating data from various sources into predefined templates.

Jinja2 was created as a successor to the original Jinja template engine. The original Jinja was inspired by the Django template system but was designed to be more flexible and extensible. However, it had some limitations, and Jinja2 was developed to address these shortcomings and provide a more powerful, feature-rich, and robust templating engine.

Jinja2 quickly gained popularity within the Python community due to its simplicity, readability, and performance. It became the default template engine for Flask, a popular web framework, which contributed to its widespread adoption.

Jinja2 offers several features and benefits that make it a valuable tool in various contexts:

Template Inheritance: Jinja2 supports template inheritance, allowing you to create a base template with common structure and blocks. Child templates can extend the base template and override specific blocks, promoting code reuse and maintainability.

Variables: You can easily insert variables into templates using double curly braces `{{ ... }}`. This enables dynamic content generation by substituting placeholders with actual data from variables.

Control Structures: Jinja2 provides control structures like `if`, `for`, and `macros`, which allow you to create conditional logic, loops, and reusable template fragments.

Filters: Filters allow you to modify variables before displaying them in templates. Filters can format dates, convert strings to uppercase, sort lists, and perform various other operations on data.

Extensibility: Jinja2 can be extended with custom filters, tests, and extensions, making it highly adaptable to specific requirements and use cases.

Jinja templating with YAML makes Ansible a powerful automation platform that enhances customizability, readability and re-usability of templates.

For more details, see: <https://jinja.palletsprojects.com/en/3.1.x/templates/>

2.3 Understanding Ansible Inventory

If you've ever worked in infrastructure management or played around with automation tools, chances are you've come across Ansible. Ansible is a powerful open-source automation tool that simplifies the management of complex IT systems. It allows users to define and deploy automation tasks through simple, human readable YAML scripts. One crucial aspect of working with Ansible is understanding the concept of an inventory.

If you want to manage your servers and applications with Ansible, there should be a way to define a list of them. This list of targets is called an inventory in Ansible. It can be a very simple static inventory – similar to a list of ingredients you want to buy today in a shop – or it can be a more complex static inventory with groups and additional variables.

The inventory can also be a dynamic inventory where you get the list from third party provider like IBM Cloud, PowerVC or Power HMC.

2.3.1 Overview of Ansible inventory

In simple terms, an inventory is a file or group of files that lists all the remote hosts (machines) that Ansible will manage. It serves as a source of truth for Ansible, allowing it to identify target systems and execute specific tasks on them. The inventory file can be formatted as plain text, INI, or YAML, and it's usually named `inventory` or `hosts`.

Inventory components

Beyond identifying remote hosts, an inventory can be much more than just a list of servers. It can also include information about groups, variables, and more. Let's dive into these key components:

- Hosts** These represent the individual machines or servers you want to manage with Ansible. Each host typically has a unique name or IP address associated with it.
- Groups** Organizing hosts into groups makes it easier to manage and perform operations on specific sets of machines. For example, you could have groups like *webservers*, *database-servers*, or even *staging* and *production* groups.
- Variables** An inventory allows you to define variables at both the host and group level. These variables can be used to customize and parameterize your playbooks. It helps in making your automation scripts more flexible and reusable.

Creating an Ansible inventory

Now that we've covered the basics, let's look at what's required to create an Ansible inventory. The following list defines the tools and definitions used to define an inventory file:

- Configuration file** Ansible requires an "ansible.cfg" file to specify the default inventory location. By default, it searches for an `inventory` file in the current directory. You can customize this location according to your project structure and needs.
- Inventory file** As mentioned before, the inventory file itself is the heart of your inventory. It can reside locally on your Ansible control node or be fetched dynamically from external sources like cloud providers, external databases, or even scripts.
- Inventory structure** It's important to understand and adhere to the inventory structure. Whether you choose the INI, YAML, or plain text format, organizing hosts and groups correctly improves the readability and maintainability of your inventory.

Using Ansible inventory

Now that we understand the components of an Ansible inventory, let us explore its uses. There are several advantages of using an inventory in your Ansible environment.

- ▶ **Defining the infrastructure**

The inventory file provides a high-level view of your infrastructure by listing all the servers and their associated attributes. This makes it easier to understand and manage the environment.
- ▶ **Targeting specific hosts or groups**

By utilizing the host and group information, you can execute Ansible tasks on subsets of machines, rather than applying them system-wide. This helps in achieving more granular control over your automation.

► **Modifying host variables**

Inventory files enable you to define host-specific variables such as IP addresses, credentials, or package versions. These variables can be accessed within playbooks and used to customize behavior based on specific hosts.

Ansible inventory plays a crucial role in managing and automating your IT infrastructure. It helps Ansible identify the target systems, organize them into groups, define variables, and execute tasks accordingly. The inventory file empowers you to automate tasks based on your specific needs by providing a clear and structured overview of your environment. Whether you are a system administrator, a DevOps engineer, or simply curious about automation, mastering the Ansible inventory is an essential skill to have in your toolkit.

Defining inventory hosts and groups

As we have already seen, your inventory defines the managed nodes you automate. Groups help you run automation tasks on multiple hosts at the same time. Once your inventory is defined, you can use patterns to select the hosts or groups you want Ansible to run against.

The simplest inventory is a single file with a list of hosts and groups. The default location for this file is `/etc/ansible/hosts` but you can specify a different inventory file at the command line using the `-i <path>` option, or in the configuration file using `inventory`.

Ansible [inventory plugins](#) support a range of formats and sources to make your inventory flexible and customizable. As your inventory expands, you may need more than a single file to organize your hosts and groups. Here are three options beyond the `/etc/ansible/hosts` file:

1. You can create a directory with multiple inventory files. See “Organizing inventory in a directory” on page 67. These can use different formats (YAML, ini, and so on).
2. You can pull inventory dynamically. For example, you can use a dynamic inventory plugin to list resources in one or more cloud providers. See 2.3.2, “Overview of dynamic inventory” on page 68.
3. You can use multiple sources for inventory, including both dynamic inventory and static files.

Organizing inventory in a directory

You can consolidate multiple inventory sources in a single directory. The simplest version of this is a directory with multiple files instead of a single inventory file. A single file gets difficult to maintain when it gets too long. If you have multiple teams and multiple automation projects, having one inventory file per team or project lets everyone easily find the hosts and groups that matter to them.

You can also combine multiple inventory source types in an inventory directory. This can be useful for combining static and dynamic hosts and managing them as one inventory. The inventory directory shown in Example 2-6 combines an inventory plugin source, a dynamic inventory script, and a file with static hosts.

Example 2-6 Inventory directory with mixed sources

```
inventory/
  openstack.yml      # configure inventory plugin to get hosts from OpenStack cloud
  dynamic-inventory.py # add additional hosts with dynamic inventory script
  on-prem           # add static hosts and groups
  parent-groups     # add static hosts and groups
```

You can target this inventory directory as follows:

```
ansible-playbook example.yml -i inventory
```

You can also configure the inventory directory in your `ansible.cfg` file.

2.3.2 Overview of dynamic inventory

If you know in advance which servers you want to manage using Ansible, static inventories are great. However, sometimes the infrastructure is so dynamic that you can not even know the names of servers, or you may know the names but it is still easier to get them from some other source. As an example, you may want to perform some operations on Virtual I/O Servers on a specific managed system, or to run some playbook on all LPARs managed by PowerVC. This is the use case for dynamic inventories.

Dynamic inventory is a way defining the list of servers to manage by using a special inventory plugin. You can find the list of plugins available to you by using the `ansible-doc` command as shown in Example 2-7.

Example 2-7 List of plugins provided by ansible-doc command

```
# ansible-doc -t inventory -l
ansible.builtin.advanced_host_list Parses a 'host list' with ranges
ansible.builtin.auto                Loads and executes an inventory plugin specified in a
YAML config
ansible.builtin.constructed        Uses Jinja2 to construct vars and groups based on
existing inventory
ansible.builtin.generator          Uses Jinja2 to construct hosts and groups from patterns
ansible.builtin.host_list          Parses a 'host list' string
ansible.builtin.ini                Uses an Ansible INI file as inventory source
ansible.builtin.script              Executes an inventory script that returns JSON
ansible.builtin.toml                Uses a specific TOML file as an inventory source
ansible.builtin.yaml                Uses a specific YAML file as an inventory source
ibm.power_hmc.powervm_inventory    HMC-based inventory source for Power Servers
openstack.cloud.openstack           OpenStack inventory source
```

We usually have several inventory plugins which are delivered with Ansible. They start with `ansible.builtin`. In this particular case there are two additional plugins provided by installed collections:

- `ibm.power_hmc.powervm_inventory` provided by the `ibm.power_hmc` collection
- `openstack.cloud.openstack` provided by `openstack.cloud` collection.

Just like with static inventories, you can use the `ansible-inventory` command to test and to show your inventory. In the case of dynamic inventories, it makes even more sense to test your configuration before you run your playbook as shown in Example 2-8.

Example 2-8 Inventory test using ansible-inventory command

```
# ansible-inventory -i hmc1.power_hmc.yml --list
{
  "_meta": {
    "hostvars": {
      "aixlpar1": {
        "ansible_host": "10.17.19.42"
      },
      "aixlpar2": {
        "ansible_host": "10.17.19.100"
      }
    }
  }
}
```

```

        "vio1": {
            "ansible_host": "10.17.19.113"
        },
        "vio2": {
            "ansible_host": "10.17.19.114"
        },
        "linux": {
            "ansible_host": "10.17.19.13"
        }
    }
},
"all": {
    "children": [
        "ungrouped",
        "P10-9080-HEX"
    ]
},
"P10-9080-HEX": {
    "hosts": [
        "aixlpar1",
        "aixlpar2",
        "vio1",
        "vio2",
        "linux"
    ]
}
}

```

From the output in Example 2-8 on page 68 we see that the dynamic inventory defined using the file `hmc1.power_hmc.yml` found several AIX, Linux and Virtual I/O Server LPARs.

If we look into the configuration file, we find information on how to connect to the HMC and filters which determine which systems and LPARs we would like to see in the output. This is shown in Example 2-9.

Example 2-9 Contents of `hmc1.power_hmc.yml`

```

# cat hmc1.power_hmc.yml
plugin: ibm.power_hmc.powervm_inventory
hmc_hosts:
  - hmc: hmc1
    user: hscroot
    password: abcd1234
system_filters:
  SystemName: 'P10-9080-HEX'
filters:
  PartitionState: 'running'

```

Example 2-10 shows a fairly common situation where you want to get a list of LPARs direct from an HMC. Using `powervm_inventory` from `ibm.power_hmc` collection you can dynamically define variables and assign servers to groups.

Example 2-10 Defining variables dynamically using `powervm_inventory`

```

# cat hmc1.power_hmc.yml
plugin: ibm.power_hmc.powervm_inventory
strict: False
hmc_hosts:
  - hmc: hmc1
    user: hscroot

```

```

    password: abcd1234
system_filters:
  SystemName: 'P10-9080-HEX'
filters:
  PartitionState: 'running'
compose:
  ansible_host: PartitionName
  ansible_python_interpreter: "/opt/freeware/bin/python3" if 'AIX' in
OperatingSystemVersion or 'VIOS' in OperatingSystemVersion else
'/QOpenSys/pkgs/bin/python3' if 'IBM i' in OperatingSystemVersion else '/usr/bin/python3'"
  ansible_user: "'root' if 'AIX' in OperatingSystemVersion or 'Linux' in
OperatingSystemVersion else 'ansible' if 'VIOS' in OperatingSystemVersion else 'qsecofr'"
groups:
  AIX: "'AIX' in OperatingSystemVersion"
  Linux: "'Linux' in OperatingSystemVersion"
  IBMi: "'IBM i' in OperatingSystemVersion"
  VIOS: "'VIOS' in OperatingSystemVersion"

```

In this example we define groups of hosts according to their operating systems. All Virtual I/O Servers will be assigned to the group VIOS, all IBM i LPARs will be assigned to the group 'IBM i', all Linux LPARs will be assigned to the group Linux and all AIX LPARs will be assigned to the group AIX.

Similar we dynamically define variables for our hosts. We want to connect to AIX and Linux LPARs as user *root* and to IBM i LPARs as user *qsecofr*. That is why we set the variable *ansible_user* depending on the LPAR's operating system. We change the path to the python interpreter based on the LPAR's operating system too.

If you use IBM PowerVC to manage your LPARs, you can use the `openstack.cloud.openstack` inventory plugin to get the list of the LPARs in PowerVC. The easiest way to use the inventory plugin is to set variables from `/opt/ibm/powervc/powervcrc` and then create an inventory file with one line in it as shown in Example 2-11.

Example 2-11 Using openstack plugin

```

# cat openstack.yml
plugin: openstack.cloud.openstack
# source /opt/ibm/powervc/powervcrc
# ansible-inventory -i openstack.yml --list

```

If you don't want to set environment variables, you can create a file named `clouds.yml` in the same directory with authentication parameters as shown in Example 2-12. It will be used by the OpenStack inventory plugin automatically.

Example 2-12 Using clouds.yml

```

# cat clouds.yml
clouds:
  powervc:
    identity_api_version: "3"
    region_name: RegionOne
    verify: false
    auth:
      auth_url: https://powervc:5000/v3
      project_name: "ibm-default"
      username: root
      password: ibmaix
      project_domain_name: Default
      user_domain_name: Default

```

You can get more information about the inventory plugins using the `ansible-doc` command:

```
# ansible-doc -t inventory ibm.power_hmc.powervm_inventory
# ansible-doc -t inventory openstack.cloud.openstack
```

It is also possible to extend the current playbook with newly provisioned hosts using dynamic inventory.

Example 2-13 shows a playbook consisting of 2 plays, one play targets the localhost and the second play targets the dynamic group "powervms".

Example 2-13 Playbook targeting localhost and dynamic group

```
---
- hosts: localhost
  tasks:
    - name: create dynamic in memory inventory
      add_host:
        name: "{{ item.ip }}"
        groups: powervms
        ansible_connection: local
      loop:
        - ip: 1.2.3.4
          name: vm01
        - ip: 2.4.5.6
          name: vm02
        - ip: 3.4.5.6
          name: vm03
- hosts: powervms
  tasks:
    - name: wait for reachability
      wait_for_connection:
        delay: 5
        timeout: 240
    - name: gather_facts
      setup:
        gather_subset: min
    - debug:
        msg: "{{ ansible_hostname }} {{ansible_default_ipv4.address}}"
```

Note that for documentation reasons we set `ansible_connection: local`, in production you may omit this line if you wish. The output from Example 2-13 is shown in Example 2-14.

Example 2-14 Playbook output

```
WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit
localhost does not match 'all'
```

```
PLAY [localhost]
```

```
*****
**
```

```
TASK [create dynamic in memory inventory]
```

```
*****
```

```
Saturday 30 September 2023 14:02:08 +0200 (0:00:00.030) 0:00:00.030 ****
Saturday 30 September 2023 14:02:08 +0200 (0:00:00.030) 0:00:00.030 ****
changed: [localhost] => (item={'ip': '1.2.3.4', 'name': 'vm01'}) => {"add_host": {"groups":
["powervms"], "host_name": "1.2.3.4", "host_vars": {"ansible_connection": "local"}},
"ansible_loop_var": "item", "changed": true, "item": {"ip": "1.2.3.4", "name": "vm01"}}
```

```

changed: [localhost] => (item={'ip': '2.4.5.6', 'name': 'vm02'}) => {"add_host": {"groups":
["powervms"], "host_name": "2.4.5.6", "host_vars": {"ansible_connection": "local"}},
"ansible_loop_var": "item", "changed": true, "item": {"ip": "2.4.5.6", "name": "vm02"}}
changed: [localhost] => (item={'ip': '3.4.5.6', 'name': 'vm03'}) => {"add_host": {"groups":
["powervms"], "host_name": "3.4.5.6", "host_vars": {"ansible_connection": "local"}},
"ansible_loop_var": "item", "changed": true, "item": {"ip": "3.4.5.6", "name": "vm03"}}

```

PLAY [powervms]

```

*****
***

```

TASK [wait for reachability]

```

*****

```

```

Saturday 30 September 2023  14:02:08 +0200 (0:00:00.085)      0:00:00.116 ****

```

```

Saturday 30 September 2023  14:02:08 +0200 (0:00:00.085)      0:00:00.116 ****

```

```

[WARNING]: Reset is not implemented for this connection

```

```

[WARNING]: Reset is not implemented for this connection

```

```

[WARNING]: Reset is not implemented for this connection

```

```

ok: [3.4.5.6] => {"changed": false, "elapsed": 5}

```

```

ok: [1.2.3.4] => {"changed": false, "elapsed": 5}

```

```

ok: [2.4.5.6] => {"changed": false, "elapsed": 5}

```

TASK [gather_facts]

```

*****

```

```

Saturday 30 September 2023  14:02:13 +0200 (0:00:05.427)      0:00:05.544 ****

```

```

Saturday 30 September 2023  14:02:13 +0200 (0:00:05.427)      0:00:05.543 ****

```

```

ok: [2.4.5.6]

```

```

ok: [3.4.5.6]

```

```

ok: [1.2.3.4]

```

TASK [debug]

```

*****

```

```

*****

```

```

Saturday 30 September 2023  14:02:14 +0200 (0:00:00.805)      0:00:06.349 ****

```

```

Saturday 30 September 2023  14:02:14 +0200 (0:00:00.805)      0:00:06.348 ****

```

```

ok: [1.2.3.4] => {
  "msg": "vm9810 192.168.98.10"
}

```

```

ok: [2.4.5.6] => {
  "msg": "vm9810 192.168.98.10"
}

```

```

ok: [3.4.5.6] => {
  "msg": "vm9810 192.168.98.10"
}

```

PLAY RECAP

```

*****

```

```

*****

```

```

1.2.3.4      : ok=3    changed=0    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0

```

```

2.4.5.6      : ok=3    changed=0    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0

```

```

3.4.5.6      : ok=3    changed=0    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0

```

```

localhost    : ok=1    changed=1    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0

```

```


```

```


```

```


```

```


```

```


```

```


```

```


```

```


```

```


```

```


```

```


```

```


```

```


```

If you want to make changes to a local inventory file instead of using an in memory inventory, you can use the *lineinfile* or *template* module to populate the inventory file. To make Ansible

use the most current file, there is a meta directive that will reread and reload the inventory. See Example 2-15 for a playbook that uses these two actions.

Example 2-15 Create dynamic inventory file

```

---
- hosts: localhost

  tasks:
    - name: create dynamic inventory
      local_action:
        module: lineinfile
        path: inventories/powervms.ini
        regexp: ^{{ item.name }}
        insertafter: "[powervms]"
        line: "{{ item.name }} ansible_host={{ item.ip }} ansible_connection=local"
      loop:
        - ip: 1.2.3.4
          name: vm01
        - ip: 2.4.5.6
          name: vm02
        - ip: 3.4.5.6
          name: vm03

    - meta: refresh_inventory

- hosts: powervms

  tasks:
    - name: wait for reachability
      wait_for_connection:
        delay: 5
        timeout: 240

    - name: gather_facts
      setup:
        gather_subset: min

    - debug:
        msg: "{{ ansible_hostname }} {{ansible_default_ipv4.address}}"

```

Example 2-16 shows the output from the playbook in Example 2-15.

Example 2-16 Output of "ansible-playbook -i playbooks/inventories/powervms.ini"

```

playbooks/dynamic2.yml -v --diff''
Script started, file is /tmp/a
Using /ansible/ALL/ansible/ansible.cfg as config file
[WARNING]: * Failed to parse /ansible/ALL/ansible/playbooks/inventories/powervms.ini with
ini plugin:
/ansible/ALL/ansible/playbooks/inventories/powervms.ini:2: Expected key=value host variable
assignment, got:
1.2.3.4
[WARNING]: Unable to parse /ansible/ALL/ansible/playbooks/inventories/powervms.ini as an
inventory source
[WARNING]: No inventory was parsed, only implicit localhost is available
[WARNING]: provided hosts list is empty, only localhost is available. Note that the
implicit localhost does
not match 'all'

```

```

PLAY [localhost]
*****
**

TASK [create dynamic inventory]
*****
Saturday 30 September 2023  14:14:36 +0200 (0:00:00.041)    0:00:00.041 ****
Saturday 30 September 2023  14:14:36 +0200 (0:00:00.040)    0:00:00.040 ****
--- before: inventories/powervms.ini (content)
+++ after: inventories/powervms.ini (content)
@@ -1,4 +1,4 @@
    [powervms]
-vm01 1.2.3.4 ansible_connection=local
+vm01 ansible_host=1.2.3.4 ansible_connection=local
    vm02 2.4.5.6 ansible_connection=local
    vm03 3.4.5.6 ansible_connection=local

changed: [localhost] => (item={'ip': '1.2.3.4', 'name': 'vm01'}) => {"ansible_loop_var":
"item", "backup": "", "changed": true, "item": {"ip": "1.2.3.4", "name": "vm01"}, "msg":
"line replaced"}
--- before: inventories/powervms.ini (content)
+++ after: inventories/powervms.ini (content)
@@ -1,4 +1,4 @@
    [powervms]
    vm01 ansible_host=1.2.3.4 ansible_connection=local
-vm02 2.4.5.6 ansible_connection=local
+vm02 ansible_host=2.4.5.6 ansible_connection=local
    vm03 3.4.5.6 ansible_connection=local

changed: [localhost] => (item={'ip': '2.4.5.6', 'name': 'vm02'}) => {"ansible_loop_var":
"item", "backup": "", "changed": true, "item": {"ip": "2.4.5.6", "name": "vm02"}, "msg":
"line replaced"}
--- before: inventories/powervms.ini (content)
+++ after: inventories/powervms.ini (content)
@@ -1,4 +1,4 @@
    [powervms]
    vm01 ansible_host=1.2.3.4 ansible_connection=local
    vm02 ansible_host=2.4.5.6 ansible_connection=local
-vm03 3.4.5.6 ansible_connection=local
+vm03 ansible_host=3.4.5.6 ansible_connection=local

changed: [localhost] => (item={'ip': '3.4.5.6', 'name': 'vm03'}) => {"ansible_loop_var":
"item", "backup": "", "changed": true, "item": {"ip": "3.4.5.6", "name": "vm03"}, "msg":
"line replaced"}

TASK [meta]
*****
*****
Saturday 30 September 2023  14:14:36 +0200 (0:00:00.709)    0:00:00.750 ****
Saturday 30 September 2023  14:14:36 +0200 (0:00:00.709)    0:00:00.750 ****

PLAY [powervms]
*****
***

TASK [Gathering Facts]
*****
Saturday 30 September 2023  14:14:36 +0200 (0:00:00.041)    0:00:00.792 ****
Saturday 30 September 2023  14:14:36 +0200 (0:00:00.041)    0:00:00.791 ****
ok: [vm03]

```



```

ok: [vm02]
ok: [vm01]

TASK [wait for reachability]
*****
Saturday 30 September 2023 14:14:38 +0200 (0:00:01.386)    0:00:02.179 ****
Saturday 30 September 2023 14:14:38 +0200 (0:00:01.386)    0:00:02.178 ****
[WARNING]: Reset is not implemented for this connection
[WARNING]: Reset is not implemented for this connection
[WARNING]: Reset is not implemented for this connection
ok: [vm02] => {"changed": false, "elapsed": 5}
ok: [vm03] => {"changed": false, "elapsed": 5}
ok: [vm01] => {"changed": false, "elapsed": 5}

TASK [gather_facts]
*****
Saturday 30 September 2023 14:14:43 +0200 (0:00:05.398)    0:00:07.577 ****
Saturday 30 September 2023 14:14:43 +0200 (0:00:05.398)    0:00:07.576 ****
ok: [vm03]
ok: [vm02]
ok: [vm01]

TASK [debug]
*****
*****
Saturday 30 September 2023 14:14:44 +0200 (0:00:00.541)    0:00:08.118 ****
Saturday 30 September 2023 14:14:44 +0200 (0:00:00.541)    0:00:08.117 ****
ok: [vm01] => {
    "msg": "vm9810 192.168.98.10"
}
ok: [vm02] => {
    "msg": "vm9810 192.168.98.10"
}
ok: [vm03] => {
    "msg": "vm9810 192.168.98.10"
}

PLAY RECAP
*****
*****
localhost                : ok=1   changed=1   unreachable=0   failed=0   skipped=0
rescued=0   ignored=0
vm01                  : ok=4   changed=0   unreachable=0   failed=0   skipped=0
rescued=0   ignored=0
vm02                  : ok=4   changed=0   unreachable=0   failed=0   skipped=0
rescued=0   ignored=0
vm03                  : ok=4   changed=0   unreachable=0   failed=0   skipped=0
rescued=0   ignored=0

```

2.4 Ansible tasks, playbooks and modules

In Ansible, a playbook consists of one or more plays in an ordered list. Each play executes part of the overall goal of the playbook by running one or more tasks. A task is the smallest unit of work that is automated using the playbook.

Each task leverages Ansible modules to achieve expected outcomes. A module is a reusable, standalone script that Ansible runs, either locally or remotely. Modules interact with the local

machine, an API, or a remote system to perform specific tasks like changing a database password or spinning up a cloud instance. We will discuss each of these in this section.

2.4.1 Creating Ansible playbooks

As mentioned before, Ansible uses the YAML syntax and playbooks are expressed in YAML format with a minimum of syntax. If you are not familiar with YAML, we did an introduction in section 2.2, “Understanding Ansible’s declarative language” on page 62. If you want additional details, see [YAML Syntax](#).

Consider installing an add-on for your text editor to help you write clean YAML syntax in your playbooks. If your preferred editor is *vi* (or *vim*), then you may add these lines to `~/.vimrc` to allow you to use **TAB** by making the **TAB** character appear as 2 blank spaces:

```
autocmd FileType yaml setlocal ts=2 sts=2 sw=2 expandtab
autocmd FileType yml setlocal ts=2 sts=2 sw=2 expandtab
```

Additional recommendations for setting up *vim* are shown [here](#).

Anatomy of an Ansible playbook

A playbook is a text file written in YAML format, and is normally saved with the extension `yml` or `yaml`.

A playbook is what drives Ansible automation. The following concepts are important to understand when building playbooks for your environment:

- ▶ **Playbook** - A text file containing a list of one or more plays to run in a specific order, from top to bottom, to achieve an overall goal.
- ▶ **Play** - An ordered list of tasks that maps to managed nodes in an inventory. This is the top level specification for a group of tasks. Defined in the play are the hosts it will execute on (the inventory) and control behaviors such as fact gathering or privilege level. Multiple plays can exist within a single Ansible playbook and may execute on different hosts.
- ▶ **Task** - The application of a module to perform a specific unit of work. A play combines a sequence of tasks to be applied, in order, to one or more hosts selected from your inventory.
- ▶ **Module** - Parametrized components or programs with internal logic, representing a single step to be done on the target machine. The modules “do” things in Ansible.
- ▶ **Plugins** - Pieces of code that augment Ansible’s core functionality. These are often provided by a manufacturer for their specific devices. Ansible uses a plugin architecture to enable a rich, flexible, and expandable feature set.

The playbook uses indentation with space characters to indicate the structure of its data. YAML does not place strict requirements on how many spaces are used for the indentation, but there are two basic rules:

- ▶ Data elements at the same level in the hierarchy (such as items in the same list) must have the same indentation.
- ▶ Items that are children of another item must be indented more than their parents

You can also add blank lines for readability.

Note: Only the space character can be used for indentation. TAB characters are not allowed.

Start of playbook

A playbook starts with a line consisting of three dashes (---) as a starting document marker and may end with three dots (...) as an end-of-document marker (the ... is optional and in practice are often omitted).

Between these markers, the playbook contains a list of plays. Each item in a YAML list starts with a single dash followed by a space. Example 2-17 shows an example playbook which is designed to capture the *oslevel* from the system.

Example 2-17 Sample playbook getting oslevel

```
---
- name: GET oslevel AIX
  hosts: all

  tasks:
    - name: Gather LPP Facts
      shell: "oslevel -s"
      register: output_oslevel

    - name: Print the oslevel
      debug:
        msg: "{{ ansible_hostname }}" has the AIX oslevel of {{ output_oslevel.stdout }}"
```

Order in plays

Please note the order in plays is always:

- pre_tasks:
- roles:
- tasks:
- handlers:

You may change the order using `include_roles`, `import_roles`, `include_tasks`, or `import_tasks`. You can also use the directive `tasks_from`: while including tasks.

The main difference between `import` and `include` is:

- All **import*** statements are preprocessed at the time playbooks are parsed.
- All **include*** statements are processed as they encountered during the execution of the playbook.

That means that **import** is static, **include** is dynamic.

Good practice is to use **import** when you deal with logical “units”. For example, separate long list of tasks into subtask files in a `main.yml`. The **include** keyword is used to make decisions based on dynamically gathered facts as shown here:

```
- include_tasks: taskrun_{{ ansible_os_family | lower }}.yml
```

For more detail on using `import` or `include` see “Including and importing other playbooks” on page 79.

Verifying playbooks

You may want to verify your playbooks to catch syntax errors and other problems before you run them. The **ansible-playbook** command offers several options for verification, including `--check`, `--diff`, `--list-hosts`, `--list-tasks`, and `--syntax-check`. Use the command shown in Example 2-18 on page 78 to check the playbook for syntax errors using `--syntax-check`.

Example 2-18 Verify playbook

```
$ ansible-playbook --syntax-check aix_oslevel.yml
playbook: aix_oslevel.yml
```

Ansible tasks

In the realm of IT automation and configuration management, Ansible shines as a powerful tool that allows you to define and execute tasks across a multitude of systems. Ansible tasks form the core building blocks of automation, enabling you to specify the desired state of your infrastructure and applications in a declarative manner. Figure 2-2 shows how tasks relate to rest of the tooling within Ansible ecosystem.

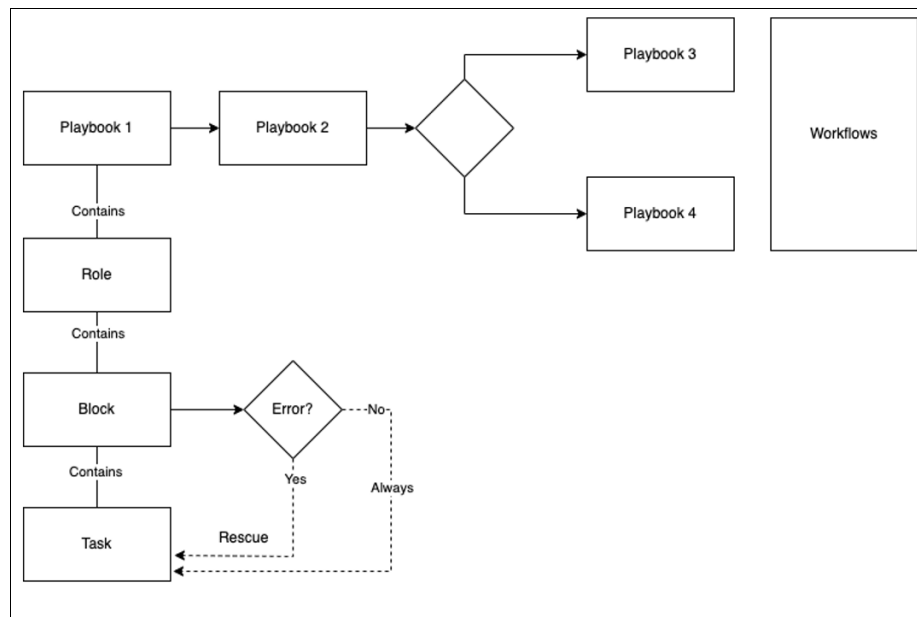


Figure 2-2 Task flow chart

Tasks are a fundamental building blocks in Ansible and there is a plethora of plugins available. Each plugin is further divided into modules and are also grouped together as collections. The plugins are of different categories and Ansible categorizes the plugins based on what they do. For example: **become** plugin allows you to become a superuser or run the function as a specific user.

The plugin ecosystem is vast and includes Ansible's own built-in plugins, as well as community plugins from contributions and vendor plugins from companies such as Cisco, IBM, and AWS for example. This [Ansible document](#) provides a list of plugins and their modules.

Writing tasks

Each task in Ansible consists of 3 components which are defined in YAML format.

The task starts with a name – which is free form text – to describe what is being done. It is the best practice to provide a good description that matches the task.

You would then call a plugin, which are of format *namespace.plugin.module* which is then followed by it's arguments or parameters.

In Example 2-19 on page 79, you see **get_url**, which is a builtin plugin in the Ansible namespace. For Ansible builtin plugins, you may omit the namespace (ansible) and plugin

(builtin). The `get_url` also shows the parameters that you need to pass such as `url`, `dest`, `mode` and `validate_certs`.

Example 2-19 Example playbook for IBM AIX that downloads and configures yum package manager, and then installs MariaDB

```

---
- name: Install MariaDB open source relational database
  hosts: ansible-vms
  tasks:
    - name: Download 'yum.sh' script
      get_url:
        url: https://public.dhe.ibm.com/aix/freeSoftware/aixtoolbox/ezinstall/ppc/yum.sh
        dest: /tmp/yum.sh
        mode: 0755
        validate_certs: False

    - name: Execute the 'yum.sh' script
      shell: /tmp/yum.sh

    - name: Install MariaDB package
      yum:
        name: mariadb-server
        state: latest

```

Including and importing other playbooks

Ansible tasks can include or import other tasks that were prebuilt. The difference between import and include is subtle, but critical to understand. The *import_tasks* are preprocessed when Ansible processes the playbook for execution. The *include_tasks* are processed when the playbook is being executed. See Example 2-20 for an excerpt from a playbook which shows the use of both the *import_tasks* and *include_tasks*.

Example 2-20 Playbook with import and include tasks

```

- name: Create filesystem when disks_configuration variables is a dictionary
  import_tasks: disks-dict2list.yml
  when: disks_configuration | type_debug == "dict"

- name: Create Filesystem when disks_configuration variables is a list
  include_tasks: file-system-creation-core.yml
  with_items:
    - "{{ disks_configuration }}"
  loop_control:
    loop_var: file_system
  when: disks_configuration | type_debug == "list"

```

It is a good practice to include tasks which apply to a specific target OS or target hardware. To get better readability of playbooks, you may use the pattern shown in Example 2-21.

Example 2-21 Using variables to target specific target attributes

```

name: get info of underlying play_hosts
  setup:
    gather_subset: min
    tags: vars
- name: gather os specific variables
  include_vars: "{{ item }}"
  with_first_found:
    - "{{ ansible_distribution }}-{{ ansible_distribution_major_version }}.yml"
    - "{{ ansible_distribution }}.yml"

```

tags: vars

Variables

Ansible uses variables to manage differences between systems. With Ansible, you can execute tasks and playbooks on multiple different systems with a single command. To represent the variations among those different systems, you can create variables with standard YAML syntax, including lists and dictionaries. You can define these variables in your playbooks, in your inventory, in re-usable files or roles, or at the command line. You can also create variables during a playbook run by registering the return value or values of a task as a new variable.

Defining Variables in playbooks

The simplest way to define variables is to put a vars section in your playbook with the names and values of your variables. See Example 2-22.

Example 2-22 Defining variables

```
vars:
  tls_dir: /etc/nginx/ssl/
  key_file: nginx.key
  cert_file: nginx.crt
  conf_file: /etc/nginx/sites-available/default
  server_name: localhost
  firewall_pkg: firewallld
  firewall_svc: firewallld
  web_pkg: httpd
  web_svc: httpd
```

Not all strings are valid Ansible variable names. A variable name can only include letters, numbers, and underscores. Python keywords or playbook keywords are not valid variable names. A variable name cannot begin with a number.

Variable names can begin with an underscore. In many programming languages, variables that begin with an underscore are private. This is not true in Ansible. Variables that begin with an underscore are treated exactly the same as any other variable. Do not rely on this convention for privacy or security. Figure 2-3 gives examples of valid and invalid variable names.

Valid variable names	Not valid
foo	*foo, Python keywords such as async and lambda
foo_env	playbook keywords such as environment
foo_port	foo-port, foo port, foo.port
foo5, _foo	5foo, 12

Figure 2-3 Ansible valid variable names

Defining variables in separate files

Ansible also allows you to put variables into one or more files, which are then referenced in the playbook using a section called *vars_files*. Let us say you want to take the preceding example and put the variables in a file named `nginx.yml` instead of putting them right in the playbook. You would replace the vars section with a *vars_files* that looks like what is shown in Example 2-23 on page 81.

Example 2-23 Defining variables in separate files

```
vars_files:
  - vars_nginx.yml
```

The *vars_nginx.yml* file would look like Example 2-22 on page 80.

Referencing variables

After you define a variable, use Jinja2 syntax to reference it. Jinja2 variables use double curly braces. For example, the expression *My amp goes to {{ max_amp_value }}* demonstrates the most basic form of variable substitution. You can use Jinja2 syntax in playbooks as seen in Example 2-24.

Example 2-24 Referencing variables

```
ansible.builtin.template:
  src: foo.cfg.j2
  dest: '{{ remote_install_path }}/foo.cfg'
```

When you want to display a debug message with a variable, then you would use a double-quoted string with the variable name embedded in double braces as you can see in Example 2-25.

Example 2-25 Referencing variables

```
- name: Display the variable
  debug:
    msg: "The file used was {{ conf_file }}"
```

Variables can be concatenated between the double braces by using the tilde operator *~*, as shown in Example 2-26.

Example 2-26 Concatenate variables

```
- name: Concatenate variables
  debug:
    msg: "Hello! Your URL is https://{{ server_name ~'.'~ domain_name }}/"
```

Registering variables

You can create variables from the output of an Ansible task with the task keyword *register*. You can use registered variables in any later tasks in your play. Remember that each Ansible module returns results in JSON format. To use these results, you create a registered variable using the *register* clause when invoking a module as shown in Example 2-27.

Example 2-27 Register variables

```
- name: Get disk informations
  ansible.builtin.shell: |
    fdisk -l
    lsblk
    df -h
  register: os_disk

- debug:
  var: os_disk
```

Example 2-28 on page 82 shows how to capture the output of the *whoami* command to a variable named *logon*.

Example 2-28 Capture whoami output

```
- name: Capture output of whoami command
  command: whoami
  register: logon
```

For more information on the use of variables see [playbooks_variables](#).

Execution control

When you execute Ansible tasks, the results are easily identified as successful or unsuccessful because by default, they return green, yellow or red.

- ▶ Green - a task executed as expected, no change was made.
- ▶ Yellow - a task executed as expected, making a change.
- ▶ Red - a task failed to execute successfully.

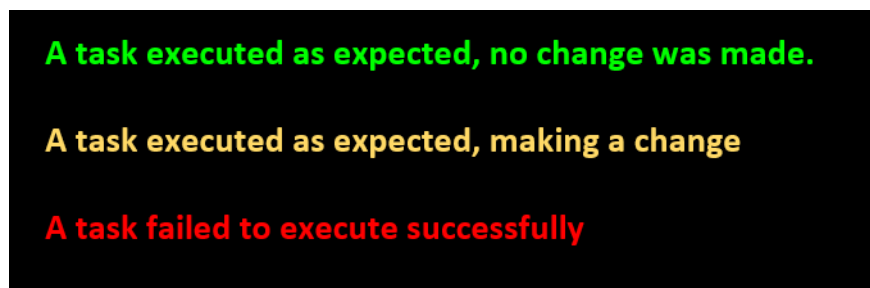


Figure 2-4 Ansible colors

2.5 Ansible roles and collections

Ansible roles are basically playbooks broken up into a known file structure. Moving to roles from a playbook makes sharing, reading, and updating your Ansible workflow easier. Users can write their own roles or they are often provided as part of collections.

2.5.1 Understanding roles in Ansible

Ansible roles are a way to organize your playbooks into reusable units that can be shared across different projects or teams. A role is essentially a collection of tasks and other files that are related to a specific task or function within your infrastructure. For example, you can create a role to install and configure IBM WebSphere® Application Server on a remote server, and then use that role in different playbooks that deploy different web applications.

To create a role, you need to follow a standard directory structure with eight main directories: *tasks*, *handlers*, *vars*, *defaults*, *meta*, *library*, *module_utils*, and *lookup_plugins*. Each directory contains a *main.yml* file that holds the relevant content for that directory. For example, the *tasks/main.yml* file contains the main list of tasks that the role executes, and the *vars/main.yml* file contains the variables associated with the role. You can also use other files and directories within each directory as needed.

To use a role in a playbook, you can either include it at the play level using the `roles` keyword, or import it at the task level using the `import_role` or `include_role` modules. You can also pass parameters to the role using the `vars` keyword or the `args` keyword. Additionally, you can specify role dependencies in the *meta/main.yml* file, which means that Ansible will automatically run those roles before the current role.

Ansible roles are a powerful feature that can help in your configuration management and automate your deployments. You can also use Ansible Galaxy to find and share roles created by other users. For further information take a look at [Ansible Roles](#).

2.5.2 Creating and structuring Ansible roles

In this part, we explore the process of creating and structuring Ansible roles using two distinct resources - the official Ansible documentation and insights from Red Hat.

- ▶ **From the Ansible Documentation:** Creating Ansible roles involves a structured approach that aligns with best practices. The official Ansible documentation offers a comprehensive guide to creating roles that are intuitive, organized, and easy to maintain. This resource provides a deep dive into the directory structure that forms the backbone of a well-structured role. To explore this approach, visit: [Official Ansible Role Directory Structure](#)
- ▶ **The Red Hat perspective:** Red Hat, a pioneer in the field of open-source technology, provides its own insights into developing Ansible roles. Their approach offers a practical and hands-on perspective that complements the official documentation. By adhering to Red Hat's methodology, you can gain a clearer understanding of how to develop Ansible roles effectively. For a step-by-step guide on creating roles the Red Hat way, navigate to: [Developing Ansible Roles the Red Hat way](#)

Crafting a role for IBM i VM deletion

In this segment, we follow a guidance to create a specialized Ansible role focused on deleting IBM i VMs. The process aligns with industry preferred practices, ensuring efficient, modular, and well-documented automation. By adhering to these principles, you will be equipped to build a robust and effective Ansible role tailored for IBM i VM deletion. The steps are:

1. Creating an Ansible role begins with establishing a well-structured directory. This ensures clarity, ease of maintenance, and efficient collaboration. To initiate the process, execute the following command, tailoring the directory path to your environment:

```
ansible-galaxy init ~/itsoxx/itsoroles/delete_vm_via_powervc
```

That command sets the foundation for your role's directory structure, adhering to the preferred practices.

2. Inside the role's directory, there are key files like *defaults/main.yml*, *meta/main.yml*, and *tasks/main.yml*. These files define the role's configuration and behavior. By meticulously editing these files, you align your role with the desired functionality. A core aspect of this step is refining the defaults values, role variables, and any specific instructions associated with your role.
3. In the *defaults/main.yml* file, you define default values for the role's variables. This enhances flexibility and customization. Example 2-29 displays a snippet showcasing how variables are defined:

Example 2-29 Defining default variables for role flexibility

```
---
# Defaults value for deleting an IBM i VM
project: ibm-default
project_domain: Default
user_domain: Default
verify_cert: false
deploy_timeout: 300
availability_zone_name: 'Default Group'
...
```

4. The core tasks of your role are defined in the `tasks/main.yml` file. This is where the automation happens. Example 2-30 is an excerpt depicting the deletion of an IBM i VM using the PowerVC module:

Example 2-30 Performing core role tasks

```
---
# Delete IBM i VM info from OpenStack
- name: Delete an IBM i VM
  os_server:
    auth:
      auth_url: https://{{ ansible_ssh_host }}:5000/v3
      username: '{{ ansible_ssh_user }}'
      password: '{{ ansible_ssh_pass }}'
      project_name: '{{ project }}'
      project_domain_name: '{{ project_domain }}'
      user_domain_name: '{{ user_domain }}'
    name: '{{ vm_name }}'
    verify: '{{ verify_cert }}'
    state: '{{vm_state}}'
    timeout: '{{ deploy_timeout }}'
  register: server_info
...
```

5. The `meta/main.yml` file is your role's calling card. It provides crucial metadata for users who can interact with your role. Example 2-31 is a snippet that showcases the role's author, description, company, supported platforms, and more:

Example 2-31 Role metadata and description

```
galaxy_info:
  author: Marcelo Avalos Del Carpio
  description: Ansible role to delete an IBM i VM via PowerVC
  company: IBM
  license: Apache-2.0
  min_ansible_version: 2.9
  platforms:
    - name: IBM i
      versions:
        - 7.2
        - 7.3
        - 7.4
  galaxy_tags:
    - powervc
    - ibmi
```

6. A well-documented role is a valuable asset. The `README.md` file provides clear instructions on how to use your role, its variables, and associated tasks. Example 2-32 is a snippet that demonstrates the structure and content of a comprehensive `README`.

Example 2-32 Comprehensive README Structure

```
delete_vm_via_powervc
=====
The Ansible role to delete an IBM i VM via PowerVC.
```

Role Variables

Variable	Type	Description
-----	-----	-----

```
| `vm_name` | str | Required. Name of the deployed vm. |
| `vm_state` | str | Action to perform (present/absent) |
```

Example Playbooks

```
-----
...
---
- name: Delete a vm
  hosts: powervc
  tasks:
    - include_role:
      name: delete_vm_via_powervc
  vars:
    vm_name: 'itso0x'
    vm_state: 'absent'
...
---
License
-----

Apache-2.0
```

Note: When publishing the created role, it is important to adhere to the formatting guidelines for the *README.md* file. Utilizing triple backticks is crucial to ensure proper readability on GitHub. For more information see [GitHub Code Blocks](#) which documents creating and highlighting code blocks:

2.5.3 Sharing and reusing roles in multiple playbooks

One of the benefits of using Ansible is that you can create reusable artifacts that can be used in different scenarios and contexts. Roles are one of the ways to achieve this. A role is a collection of related tasks, variables, defaults, handlers, and other Ansible components that can be applied to a group of hosts or a specific playbook.

Roles allow you to organize your automation work into smaller, more manageable units that can be easily shared and reused. You can use roles to abstract common functionality, such as installing a web server or updating a database, and then use them in multiple playbooks or even multiple times within one playbook.

To use roles in your playbooks, you need to follow a defined directory structure that Ansible recognizes. Each role must have at least one of the following directories:

tasks:	The main list of tasks that the role executes.
handlers:	Handlers, which may be used within or outside this role.
library:	Modules, which may be used within this role.
files:	Files that the role deploys.
templates:	Templates that the role deploys.
vars:	Other variables for the role.
defaults:	Default variables for the role. These variables have the lowest priority of any variables available, and can be easily overridden by any other variable, including inventory variables.
meta:	Metadata for the role, including role dependencies.

Each directory must contain a *main.yml* file (or *main.yaml* or *main*) that contains the relevant content for that directory. You can also use other YAML files in some directories to organize your tasks or variables better.

Note: For more insights into sharing and reusing roles across multiple playbooks, you can explore the following resources:

- ▶ Ansible Playbook Reuse Guide:
https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_reuse.html
- ▶ Sharing and Reusing Roles in Ansible:
https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html
- ▶ Playbook Reuse with Ansible Roles:
https://runebook.dev/en/docs/ansible/user_guide/playbooks_reuse_roles

2.5.4 Role dependencies and role-based variables

Role dependencies and role-based variables are important concepts in Ansible. In this section we explain what they are, how they work, and why they are useful.

Role dependencies

Role dependencies let you automatically pull in other roles when using a role, ensuring that the target computer is in a predictable condition before running your tasks. For example, you can have a common role that installs some packages and updates the system, and you want to run it before any other role.

To define role dependencies, you need to create a *meta/main.yml* file inside your role directory with a dependencies block. Example 2-33 shows an example of dependencies as follows:

Example 2-33 Role dependencies and conditional configuration in YAML

```
dependencies:
  - role: common
  - role: sshd
    enable_sshd: false
    when: environment == 'production'
```

This means that before running the current role, Ansible will first run the common role and then the sshd role, but only if the environment variable is set to **'production'**. You can also pass variables to the dependent roles using the same syntax.

Role dependencies are executed before the roles that depend on them, and they are only executed once per playbook run. If two roles state the same one as their dependency, it is only executed the first time.

For example, if you have three roles: role1, role2 and role3, and both role1 and role2 depend on role3, the execution order is as follows:

role3 → **role1** → **role2**

You can override this behavior by setting *allow_duplicates: true* in the *meta/main.yml* file of the dependent roles. This will make Ansible run the role every time it is listed as a dependency. For example, if you set *allow_duplicates: true* for role3, the execution order can be:

role3 → **role1** → **role3** → **role2**

Role dependencies are a feature of Ansible that can help you reuse your roles and simplify your playbooks. However, use them with caution and avoid creating circular dependencies or complex dependency chains that can make your roles hard to maintain and debug.

Note: For more information about role dependencies, you can refer to the official Ansible documentation or the tutorial as follows: [Ansible reuse roles](#) or [Role dependencies](#)

2.5.5 Using collections

Collections are a way to package and distribute Ansible content, such as playbooks, roles, modules, and plugins. Collections can help you organize your Ansible projects and share them with other users or teams.

What are collections

A collection is a directory of files that follows a specific structure and contains a *MANIFEST.json* file that defines its metadata. A collection can include any type of Ansible content, such as:

- Playbooks: YAML files that define tasks to run on hosts.
- Roles: Reusable units of Ansible content that can include tasks, variables, templates, files, and handlers.
- Modules: Python files that execute tasks on hosts and return information to Ansible.
- Plugins: Python files that extend Ansible's functionality, such as lookup, filter, inventory, callback, and strategy plugins.

A collection can also have dependencies on other collections, which are specified in a *meta/requirements.yml* file. This allows you to reuse existing content from other sources and avoid duplication.

How to use collections

There are several ways to use collections in Ansible, depending on your needs and preferences. Here are some of the common methods:

- ▶ Installing collections from a distribution server: You can use the **ansible-galaxy** command to install collections from a server like Ansible Galaxy or a Pulp 3 Galaxy server. You can also use a requirements file to install multiple collections at once.
- ▶ Installing collections from source files: You can use the **ansible-galaxy** command to install collections from local source files, such as a git repository or a tarball.
- ▶ Using collections in a playbook: You can reference collection content by its fully qualified collection name (FQCN), which consists of the namespace, the collection name, and the content name. For example: *my_namespace.my_collection.my_module*. You can also use the `collections` keyword in your playbook or role to simplify the module names and avoid typing the FQCN every time.
- ▶ Using collections in roles: You can use the `collections` keyword in your role's *meta/main.yml* file to define which collections your role should search for unqualified module and action names.

What are the benefits of collections

Collections offer several advantages for Ansible users and developers, such as:

- Modularity:** Collections allow you to group related content together and separate it from other content. This makes it easier to manage, maintain, and update your Ansible projects.
- Reusability:** Collections enable you to reuse existing content from other sources and avoid reinventing the wheel. You can also share your own collections with others through distribution servers or source control.
- Compatibility:** Collections help you avoid conflicts between different versions of Ansible content. You can specify the minimum Ansible version and the required collections for your collection to work properly.
- Customization:** Collections allow you to customize your Ansible environment by adding new modules and plugins that suit your needs. You can also override existing content by using a higher precedence collection.

Note: For more detailed information, please refer to the following websites:

- ▶ Ansible User Guide on Using Collections:
https://docs.ansible.com/ansible/latest/user_guide/collections_using.html
- ▶ Ansible Galaxy Guide on Creating Collections:
https://galaxy.ansible.com/docs/contributing/creating_collections.html
- ▶ Ansible Community General Collection on GitHub:
<https://github.com/ansible-collections/community.general>

2.6 Preferred practices for playbook and role design

When designing playbooks and roles, it is important to follow some preferred practices to ensure the quality, readability, and maintainability of your code. Here are some suggestions:

- ▶ Use consistent naming conventions for your files, variables, tasks, and handlers. Follow the Ansible style guide for more details: [Ansible documentation style guide](#)
- ▶ Use tags to group related tasks and allow for selective execution of your playbooks. For example, you can use tags to run only the tasks related to installing packages or configuring services.
- ▶ Use variables to store values that can change depending on the environment, such as hostnames, ports, passwords, etc. Avoid hard-coding these values in your tasks. Use the **ansible-vault** command to encrypt sensitive variables if needed.
- ▶ Use roles to organize your tasks into reusable units of functionality. Roles can also include files, templates, variables, defaults, handlers, and meta information. Use the **ansible-galaxy** command to create and manage roles.
- ▶ Use **include** and **import** statements to modularize your playbooks and roles. **Include** statements allow for dynamic inclusion of tasks or roles based on conditions or variables. **Import** statements allow for static inclusion of files or roles at parse time.
- ▶ Use conditionals, loops, filters, and plugins to add logic and flexibility to your tasks. For example, you can use conditionals to check the state of a system before performing an action, loops to iterate over a list of items, filters to manipulate data, and plugins to extend the functionality of Ansible.
- ▶ Use facts and registered variables to capture information from the hosts and use it in your tasks. Facts are variables that are automatically gathered by Ansible when running a

playbook. Registered variables are variables that are created by registering the output of a task.

- ▶ Use error handling techniques to deal with failures and unexpected situations. For example, you can use the `ignore_errors`, `failed_when`, `changed_when`, and `rescue` or `always` keywords to control the behavior of your tasks when an error occurs.

2.6.1 Writing modular and reusable playbooks

One of the benefits of using Ansible is that it allows you to write playbooks that are modular and reusable. This means that you can avoid repeating the same tasks or variables in different playbooks, and instead use existing modules, roles, or include statements to reuse code. This makes your playbooks more maintainable, scalable, and reliable.

Some of the ways to write modular and reusable playbooks are:

Use modules: Modules are reusable pieces of code that perform specific tasks, such as installing packages, creating files, or managing services. Ansible has hundreds of built-in modules that you can use in your playbooks, or you can write your own custom modules. Modules can take parameters to customize their behavior, and return information that you can use in other tasks or templates.

Use roles: Roles are collections of tasks, variables, files, templates, and handlers that are organized by a specific functionality or purpose. Roles allow you to group related tasks together and reuse them in multiple playbooks. Roles also support dependencies, which means that you can specify other roles that need to be executed before or after a given role.

Use include statements: Include statements allow you to dynamically load tasks, variables, handlers, or roles from another file or directory. This way, you can split your playbooks into smaller and more manageable files, and avoid duplication of code. You can also use conditional statements or loops to control when and how many times an include statement is executed.

Note: For more information, visit the following resources:

- ▶ Ansible Documentation: <https://docs.ansible.com/ansible/latest/index.html>
- ▶ Ansible Best Practices: https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html
- ▶ Ansible Modules: https://docs.ansible.com/ansible/latest/collections/index_module.html

2.6.2 Using Ansible Galaxy for role management

Ansible Galaxy serves as a shared repository for Ansible roles and collections, offering prepackaged units of work that can be easily shared and reused within the Ansible community. Below, we explore the process of utilizing Ansible Galaxy to discover, install, and employ roles and collections for your automation endeavors. Additionally, we explore creating your own roles and collections and uploading them to the Galaxy platform.

Discovering and installing roles and collections from Galaxy

To uncover roles and collections on Galaxy, the `ansible-galaxy search` command can be employed, incorporating filters such as author, tag, platform, or keyword. For instance, to locate roles pertaining to Linux by the author "itso," execute:

```
ansible-galaxy search linux --author itso --role
```

Example 2-34 displays a list of matching roles.

Example 2-34 Matching roles in the Ansible Galaxy search results

Found 2 roles matching your search:

Name	Description
-----	-----
itso.linux_common	Common tasks for Linux on Power.
itso.linux_application	Application deployment on Linux for Power.

For more role details, the `ansible-galaxy info` command provides comprehensive information:

```
ansible-galaxy info itso.linux_common
```

To install a role from Galaxy, the `ansible-galaxy install` command is used:

```
ansible-galaxy install itso.linux_common
```

Applying roles and collections in playbooks

To employ a role in a playbook, adding it to the roles section is sufficient, Example 2-35 shows an example.

Example 2-35 Implementing 'itso.linux_common' role in Ansible playbook

```
- hosts: all
  roles:
    - itso.linux_common
```

Variables can be passed using the `vars` or `vars_files` keywords. For instance see Example 2-36.

Example 2-36 Applying 'itso.linux_common' role with custom variables in Ansible playbook

```
- hosts: all
  roles:
    - role: itso.linux_common
      vars:
        var_name: value
```

Using collections in playbooks requires specifying the full namespace shown in Example 2-37.

Example 2-37 Utilizing 'community.general' module in Ansible playbook task

```
- hosts: all
  tasks:
    - name: Task using community.general module
      community.general.module_name:
        param1: value
        param2: value
```

Uploading roles and collections to Galaxy

Creating a role begins with the `ansible-galaxy init` command:

```
ansible-galaxy init myrole
```

For collections, the `ansible-galaxy collection init` command is used:

```
ansible-galaxy collection init mynamespace.mycollection
```

Editing files such as `README` and `metadata` is essential. Uploading requires an account on the Galaxy website and an API key. Building and uploading involve commands shown in Example 2-38.

Example 2-38 Building and publishing an Ansible role to Ansible Galaxy

```
$ ansible-galaxy role build myrole  
$ ansible-galaxy role publish myrole-1.0.0.tar.gz --api-key <API_KEY>
```

Note: Ansible Galaxy stands as a valuable asset for harnessing the potential of Ansible automation through its extensive collection of roles and collections. Empowered by the `ansible-galaxy` command line tool, users gain the ability to explore, install, and contribute to these automation components, fostering collaboration and efficiency. For more in-depth insights, you can refer to the following resources:

- ▶ Ansible Galaxy's guide on creating roles:
https://galaxy.ansible.com/docs/contributing/creating_role.html
- ▶ Ansible Galaxy's user guide:
https://docs.ansible.com/ansible/latest/galaxy/user_guide.html

2.7 Versioning and documenting playbooks and roles

One of the preferred practices for Ansible is to use roles to organize and reuse your automation content. Roles are self-contained units of Ansible automation that can be shared and used by multiple playbooks. Roles can also include custom modules, which are small programs that perform actions on remote hosts or on their behalf.

In this section we explore how to version and document your playbooks and roles, so that you can maintain them easily and collaborate with others effectively.

Versioning playbooks and roles

Versioning plays a crucial role in software development projects, including Ansible. It serves as a vital mechanism for monitoring code changes, maintaining historical records, and establishing distinct versions or branches for various project needs. Just as in any software development context, versioning in Ansible is of utmost importance for effective collaboration and project management.

To version your playbooks and roles, you should use a version control system (VCS) such as Git, which is a popular and widely used tool for managing code repositories. Git allows you to create snapshots of your code at any point in time, called commits, and to switch between different versions or branches of your code, called checkout.

Using Git, you can also push your code to a remote repository, such as GitHub or Bitbucket, where you can store it safely and share it with others. You can also pull code from a remote repository to update your local copy with the latest changes.

To effectively utilize Git with Ansible, consider these steps:

- ▶ Initialize a Git repository in your project directory housing playbooks and roles.
- ▶ Incorporate a *.gitignore* file to exclude irrelevant files from versioning, such as temporary files or sensitive data.
- ▶ Add and commit your playbooks and roles to the repository, utilizing descriptive messages for clarity on changes.
- ▶ Create branches for distinct features or environments, such as development, testing, or production.
- ▶ Merge branches when ready to integrate changes into the primary branch, often labeled master.
- ▶ Push your code to a remote repository, ensuring access and collaboration from anywhere.
- ▶ Pull code from the remote repository to sync your local copy with the latest updates.

Common scenarios when using git with Ansible

How you get started using git depends on whether you will be creating a new source code repository from scratch, contributing to an existing repository, or creating a fork or branch of an existing repository to create an alternate repository.

Creating a git repository from scratch

If you have a collection of Ansible playbooks and associated files that you want to start managing using git version control, you can create a new repository with the **'git init'** command. This will create a new local git repository, which you can then **'push'** to a remote repository to allow collaboration and access to remotely stored copies of your code. Sample commands for creating a repository with **'git init'** are shown in Example 2-39. It is recommended that each repository have a *README.md* file describing the repository.

Example 2-39 Create local repo with git init

```
% ls
ansible.cfg inventory update-hosts.yml
% echo "# My first repo" > README.md
% git init
Initialized empty Git repository in /<pathname>/.git/
% git add README.md ansible.cfg inventory update-hosts.yml
% git commit -m "first commit"
[main (root-commit) b7bb240] first commit
 4 files changed, 289 insertions(+)
 create mode 100644 README.md
 create mode 100644 ansible.cfg
 create mode 100644 inventory
 create mode 100644 update-hosts.yml
% git branch -M main
```

To push the local repository to GitHub, you should first create a new repository at <https://github.com/new>. Your repository can be either private or public. The commands in Example 2-40 shows how you would push your local repository to a remote GitHub you have created named *'my-first-ansible-repo'* as the GitHub user *'username'*.

Example 2-40 Push local repo to remote location

```
% git remote add origin https://github.com/username/my-first-ansible-repo.git
% git push -u origin main
Username for 'https://github.com': username
Password for 'https://username@github.com':
```

```
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 10 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 1.55 KiB | 1.55 MiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/username/my-first-ansible-repo.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

Please note that the password for the **'git push'** command should be a user token created at <https://github.com/settings/tokens>.

Now that you have your code stored in both a local and remote git repository, any changes to existing files or newly created files can be added to the repositories with the git **'status'**, **'add'**, **'commit'** and **'push'** commands. A sample session is shown in Example 2-41 where a new playbook named *'install-pkg.yaml'* and the updated playbook *'update-hosts.yaml'* are committed to the repository and pushed to the previously created remote repository.

Example 2-41 Commit changes to remote repo

```
% git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
   modified:   update-hosts.yaml

Untracked files:
  (use "git add <file>..." to include in what will be committed)
   install-pkg.yaml

no changes added to commit (use "git add" and/or "git commit -a")

% git add update-hosts.yaml install-pkg.yaml
% git commit -m "my second commit"
[main 50ce021] my second commit
 2 files changed, 1 insertion(+), 1 deletion(-)
 create mode 100644 install-pkg.yaml
% git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
% git push
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 10 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 345 bytes | 345.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/username/my-first-ansible-repo.git
 b7bb240..50ce021  main -> main
```

Contributing to an existing GitHub repository

You may be part of a team that has a Git repository where multiple team members contribute code to the repository. In this case you would get started by first cloning the repository to your local machine. Example 2-42 shows how you would clone a repository called *'my-team-repo'* to your local machine.

Example 2-42 Cloning a remote repo with git clone

```
$ git clone https://github.com/username/my-team-repo.git
Cloning into 'my-team-repo'...
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 10 (delta 2), reused 10 (delta 2), pack-reused 0
Receiving objects: 100% (10/10), done.
Resolving deltas: 100% (2/2), done.
$ cd my-team-repo/
$ ls
ansible.cfg  install-pkg.yaml  inventory  README.md  update-hosts.yaml
```

Modifications and additions to the local copy of the repository can be committed to the remote repository by the methods shown in Example 2-41 on page 93. It is good practice to regularly run **'git pull'** as shown in Example 2-43 on your local repository to pull down other contributors' changes.

Example 2-43 Using git pull to keep repository up to date

```
$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 3 (delta 2), reused 3 (delta 2), pack-reused 0
Unpacking objects: 100% (3/3), 269 bytes | 134.00 KiB/s, done.
From https://github.com/username/my-team-repo
   50ce021..1d4978e  main       -> origin/main
Updating 50ce021..1d4978e
Fast-forward
   inventory | 1 +
   1 file changed, 1 insertion(+)
```

Creating a fork or branch of existing repository

Forking a repository creates a copy of that repository under your account so you can modify code from the original without affecting the original repository. For example you may want to fork your own version of a repository like *ansible-power-aix* to experiment with code changes specific to your environment.

Branches of a repository can be used to support different versions of an application, or to create new features.

Branches can later be merged with the original repository, if appropriate, by a **'git merge'**, usually initiated by a pull request submitted to the original repository administrator. The pull request will inform the repository administrator that there are committed changes to a branch that you wish to merge with the original repository. The administrator may or may not merge the committed changes of the branch to the original repository.

Note: For comprehensive guidance on using Git with Ansible, consult this resource: Ansible Best Practices - Content Organization: https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html#content-organization

Documenting Playbooks and Roles

Documentation is a pivotal means of describing code purpose, functionality, and usage instructions. This practice holds significance across software development, and Ansible is no exception.

To adeptly document your Ansible playbooks and roles, comments and metadata files serve as vital tools. Comments, while ignored by Ansible, provide human-readable context. Metadata files, structured as YAML files, house key-value pairs elucidating code characteristics.

For effective documentation via comments and metadata files, consider these steps:

- ▶ Utilize comments within playbooks and roles to explain tasks, handlers, variables, or templates, alongside assumptions and dependencies.
- ▶ Employ metadata files in roles to provide information such as role name, description, author, license, platforms, dependencies, tags, variables, examples, and more.
- ▶ Use the **ansible-doc** command to generate documentation from metadata files in HTML or plain text format.
- ▶ Utilize the **ansible-galaxy** command to upload roles to Ansible Galaxy, a public repository of roles which can be downloaded by anyone.

Note: For more information and detailed guidance, please explore the following resources:

- ▶ Ansible Best Practices - Task and Handler Organization for a Role https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html#task-and-handler-organization-for-a-role
- ▶ Ansible Developing Modules - Documenting Modules https://docs.ansible.com/ansible/latest/dev_guide/developing_modules_documenting.html
- ▶ Ansible Galaxy - Creating a Role https://galaxy.ansible.com/docs/contributing/creating_role.html

2.8 Testing and validating playbooks and roles

Testing and validating playbooks and roles is an important part of Ansible automation. It ensures that your playbooks and roles work as expected and avoid errors or failures when deploying them to your target hosts. In this section, we cover some basic guidelines for writing, testing and validating your playbooks and roles using Ansible tools and preferred practices.

Testing playbooks and roles

Testing your playbooks and roles is essential to ensure that they work as intended and do not cause any unexpected or undesired effects on your target hosts. There are several ways to test your playbooks and roles using Ansible tools:

Check mode

You can use the `--check` flag when running a playbook or a role to see what changes can be made without actually applying them. This can help you spot any errors or inconsistencies in your code before executing it. Check mode does not run scripts or commands, so you need to disable it for some tasks using `check_mode: false`.

Modules

You can use certain modules that are useful for testing, such as `assert`, `fail`, `debug`, `uri`, `shell` or `command`. These modules can help you verify the state or output of your target hosts, check for certain conditions or values, display messages or variables, or run arbitrary commands or scripts.

Linting

You can use **ansible-lint** to check your playbooks and roles for syntax errors, formatting issues, best practices violations, or potential bugs. Ansible-lint can also be integrated with other tools such as editors, IDEs, CI/CD pipelines, or pre-commit hooks.

Integration testing

You can use integration testing to test your playbooks and roles against different configurations or environments. For example, you can use Vagrant, Docker, or cloud VMs to create isolated test hosts with different operating systems or versions. You can also use inventory files or variables to define different parameters for your test hosts, such as package versions or service states.

Validating playbooks and roles

Validating your playbooks and roles is the final step before deploying them to your production hosts. It ensures that your playbooks and roles perform the desired actions and produce the expected results on your target hosts. There are several ways to validate your playbooks and roles using Ansible tools:

Dry run

You can use the `--diff` flag when running a playbook or a role in check mode to see the differences between the current state and the desired state of your target hosts. This can help you verify that the changes are correct and complete before applying them.

Idempotence

You can run your playbook or role multiple times on the same target host to check that it is idempotent. This means that it cannot make any changes on subsequent runs unless the state of the host has changed externally. Idempotence ensures that your playbook or role is consistent and reliable.

Verbose

The verbose option allows you to see more details about what Ansible is doing when it executes your playbooks and roles. You can use different levels of verbosity, from `-v` to `-vvvv`, to increase the amount of information displayed. The verbose option can help you debug your Ansible code, identify errors, and check the results of your tasks.

Notifications

You can use handlers to trigger notifications when certain tasks make changes on your target hosts. For example, you can use handlers to restart a service, reload a configuration file, or send an email alert. Handlers can help you validate that your changes have taken effect on your target hosts.

Reports

You can use callbacks or plugins to generate reports on the execution of your playbooks or roles. For example, you can use callbacks to display statistics, summaries, logs, or graphs of your playbook or role runs. You can also use plugins to send reports to external systems or services, such as Slack, email, or webhooks. Reports can help you validate that your playbooks or roles have run successfully and without errors.

Note: For more detailed insights about Testing and Validating playbooks and roles, please refer to the following sources:

- ▶ Five Questions for Testing Ansible Playbooks and Roles:
<https://www.ansible.com/blog/five-questions-testing-ansible-playbooks-roles>
- ▶ Ansible Testing Strategies:
https://docs.ansible.com/ansible/latest/reference_appendices/test_strategies.html
- ▶ Introduction to Ansible Playbooks:
https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_intro.html



Getting started with Ansible

Up to this point, we have discussed the advantages of using automation and specifically have seen that Ansible can be an excellent choice for automation because it is a single solution that helps automate a large number of environments including networks, cloud resources and servers. This chapter is designed to show you how to get started on your automation journey with Ansible in your IBM Power environments.

We discuss the different architectures that you might want to consider when designing your Ansible automation environment, depending on your business requirements. We also provide guidance on choosing the right server (or it might be more than one server) to be the Ansible controller and then we show you how to install Ansible controller in your IBM Power environment and show you how to prepare your different IBM Power based LPARs to be Ansible clients.

The following topics are discussed in this chapter:

- ▶ Architecting your Ansible environment
- ▶ Choosing the Ansible controller node
- ▶ Installing your Ansible control node
- ▶ Preparing your systems to be Ansible clients

3.1 Architecting your Ansible environment

There are several stages of Ansible adoption and we talk about them in this section. Depending on your Ansible experience, your environment size, and your automation expectations, you may want to start small at the beginning and then grow through different stages as you expand your automation platform. However, you might also choose to go directly to architecting and developing a fully scalable Ansible environment which will allow you to support hundreds of developers and users. This section helps you understand the different architectures that you can choose for your Ansible environment based on your business requirements.

3.1.1 Start simple – Ansible Core and Ansible Community

If you have never worked with Ansible or are just starting your automation journey with IBM Power and Ansible, do it as simply as possible. Do not over complicate things – keep it simple. The most important outcome on this stage is to get your first tasks automated and to provide a positive experience for your users.

In this simple case the architecture for your Ansible installation may look like Figure 3-1.

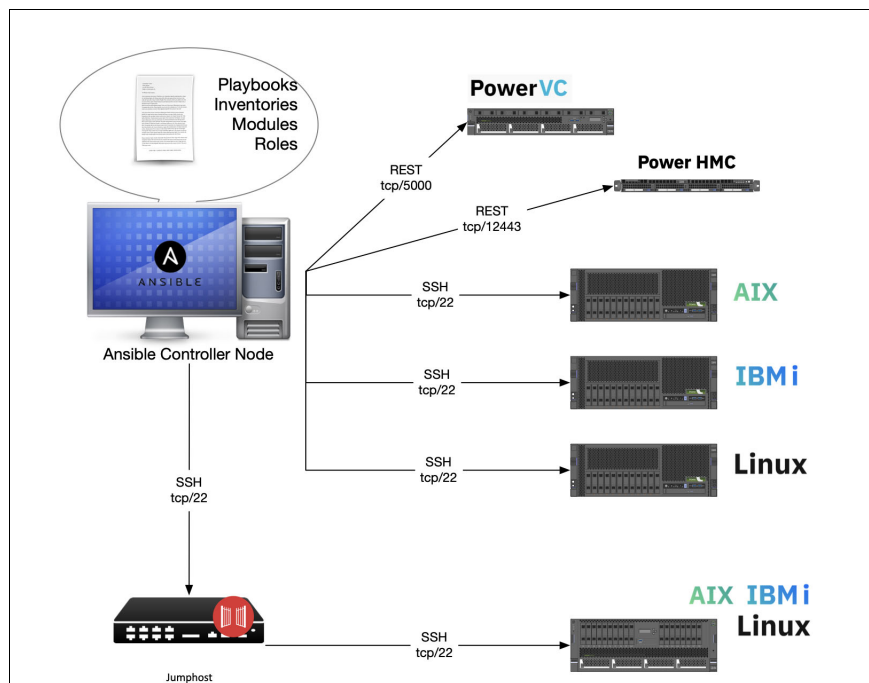


Figure 3-1 Using Ansible Core or Ansible Community to manage your IBM Power estate

Your first step is to choose the right server for your Ansible controller node. The Ansible controller node is a server where you install Ansible and develop your playbooks. There are only two requirements for this controller node server:

1. You can install Ansible on it
2. The server has SSH access to the IBM Power servers and other devices that you want to manage.

Satisfying the first requirement is very easy because Ansible can be installed on any supported operating system which runs on IBM Power. For more information about choosing

an operating system for Ansible controller node, see section 3.2, “Choosing the Ansible controller node” on page 117.

Ansible primarily works using SSH connections to the managed devices although there are some exceptions. For example, in the IBM Power world, for both IBM PowerVC and the IBM Power Hardware Management Console (HMC) Ansible works by using the REST APIs provided by those products.

If you plan to automate your Linux, AIX or IBM i LPARs, you will need SSH access to them. Ansible does support the use of SSH gateways (jump servers or bastion hosts) for access to devices with additional security requirements.

Managing your automation playbooks

Before starting to automate your first task, you must make one more decision – where will you save your work? If you are the only playbook developer at this time, your home directory is the perfect place. If you have two or more people who will work with playbooks, you may want to create a directory which can be accessed by everyone.

Even in the simplest case, you should consider using some sort of source control system like Git to track the history of changes in the playbook. While it is not necessary, it is always a good decision and it will make development more transparent and traceable. Even the best of us have problems remembering what and why they did something a year or two ago.

After choosing your Ansible controller node and deciding where to save your future work, it is time to install Ansible. You will find the steps to install Ansible on your preferred operating system in 3.3, “Installing your Ansible control node” on page 117.

Get started

Once you have installed Ansible on your controller node, you are ready to automate your first task. Consider the following when you choose this first task:

- ▶ Choose a simple task which you do regularly.
- ▶ Keep it simple. Avoid perfectionism.
- ▶ Automate step by step. Don't try to develop the whole playbook at once.
- ▶ Measure the outcome. How much time did you save?
- ▶ Speak to your colleagues about your success. Spread the word about Ansible.

Tip: In this simple architecture you will use either Ansible Core or Ansible Community for your Ansible controller. Both of these products are supported by the Ansible community and not supported by any vendor. While it is easy to start with them, you need to consider if you will need a better and more reliable support option. As you move into more production environments you will want to consider using a vendor supported product such as Red Hat Ansible Automation Platform which provides support for Ansible. Ansible Automation Platform is discussed in section 3.1.2, “Scaling up – Ansible Automation Platform” on page 101.

3.1.2 Scaling up – Ansible Automation Platform

When you are starting out and architecting the environment for one or two teams, you may only need to use one of the components of Ansible Automation Platform – the Ansible Automation Controller (formerly known as Ansible Tower). Ansible Automation Controller can be installed on IBM Power or on any x86 server.

If you have experience with base Ansible Core and have several team members, you may want to go the next level and use Ansible Automation Platform. This is especially true if you

have several administrators who develop the playbooks and a number of users who only run the playbooks.

Ansible Automation Platform provides Role-Based Access Control (RBAC) for your Ansible environment which enables you to define the rights of your automation users – which playbooks they can run and what parameters they may use in those playbooks. It also defines the rights of automation developers, defining which projects they are working on.

In addition, your users will be delighted with the nice graphical user interface of Ansible Automation Platform, which removes the complexity of running single commands and remembering the appropriate parameters to use in their Ansible playbook.

Another feature of Ansible Automation Platform is support for execution environments. Execution environments are container images that make it possible to incorporate system-level dependencies and collection-based content. Each execution environment allows you to have a customized image to run jobs, and each of them contain only what you need when running the job, nothing more. Using the execution environments you can predefine Ansible environments for automation users so they no longer have problems managing the different parameters presented by multiple versions of modules, roles, and collections. You effectively define the “single source of truth” for your automation.

Ansible Automation Platform is fully supported by Red Hat. If you are architecting an automation platform for an enterprise, support for Ansible is one of the most important consideration, and you need to clarify your requirements before starting. Ansible Core and Ansible Community are open-source projects which come with only community support. Even if it is very easy to install Ansible on every imaginable operating system, the only support you can get for it is support from the community through the projects code repositories on GitHub. What would be the impact if your enterprise automation solution has an issue and you are dependent on the community members, many of whom do not understand IBM Power, to find a solution to your problems. That is why it makes sense to have a support contract with Red Hat for any enterprise automation environment.

Currently, Red Hat supports Ansible Automation Platform on IBM Power running Red Hat Enterprise Linux as a “Technology Preview”. This means that Red Hat can’t guarantee the stability of all features on IBM Power, but will attempt to resolve any issues that customers may experience. More about technology preview support can be found in this [Red Hat document](#). Similar to IBM, Red Hat does not do any forward-looking statements and so has not stated when Ansible Automation Platform on IBM Power will be fully supported, but you can expect it in the near future. When running Ansible Automation Platform on Red Hat Enterprise Linux, you can manage systems running any Linux distribution, AIX or IBM i as supported use cases, however Red Hat Ansible Automation Platform can only be installed on Red Hat Enterprise Linux.

Ansible Automation Controller requirements

To run Ansible Automation Controller, you need Red Hat Enterprise Linux 8.6 or newer and at least 16GB RAM. The actual memory requirement for the automation controller depends on the maximum number of hosts that the system will be expected to configure in parallel. This is managed by the *forks* configuration parameter in the job template or system configuration. You will need to increase memory as you expand the number of forks to support additional systems. Red Hat recommends 1 GB of memory per 10 forks, and 2 GB for the automation controller services. If you set the forks parameter to 400, then the automation controller requires 42 GB of RAM.

You will need at least 40GB of free space in */var* on your server to proceed with the installation. You may need even more if you have a lot of playbooks and different execution

environments. To get a suitable performance for your installation, the storage utilized should be capable of a minimum of 1500 IOPS.

When using Automation Controller, you can store all your playbooks locally on the server where the Automation Controller is installed by creating one or more subdirectories under `/var/lib/awx/projects/` and copy your playbooks there. While this is easy if you have one automation developer, as you add additional automation developers – all working on different projects – you may need several different directories and you will need to manage file permissions for all of the users and projects. A flat filesystem containing files does not have any source control mechanism built in, so we recommend the use of Github, Gitlab or some similar software to store your playbooks source code centrally and enable collaboration between automation developers. Figure 3-2 shows the use of Ansible Automation Controller managing an IBM Power infrastructure.

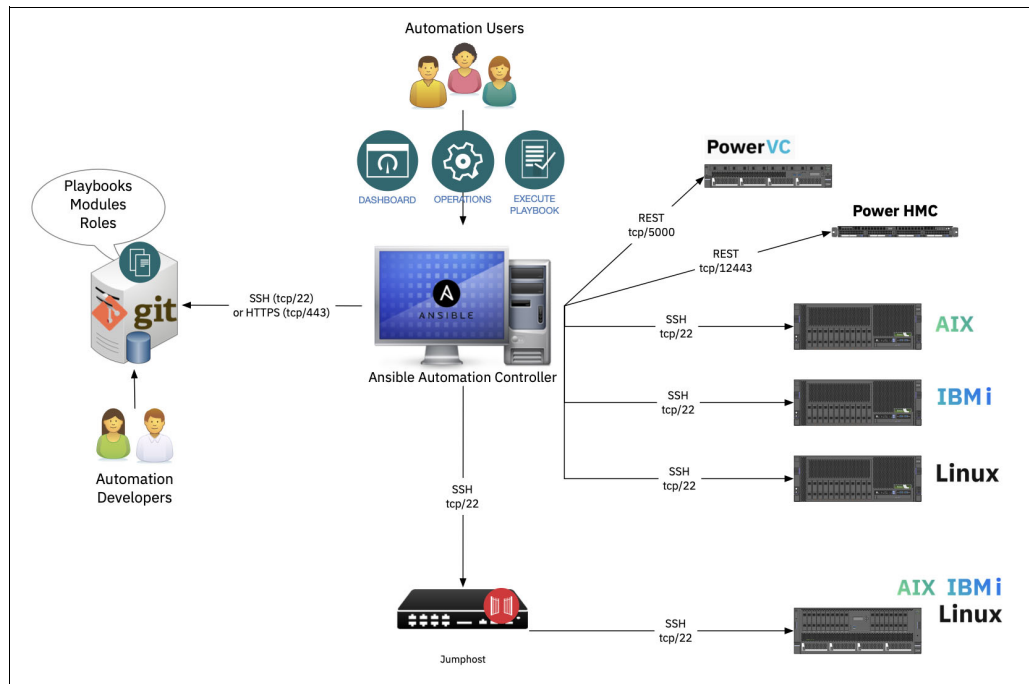


Figure 3-2 Using Ansible Automation Controller to manage IBM Power infrastructure

As you can see in Figure 3-2, the architecture is almost the same as the one using Ansible Core. However, it enables you to support more developers and users of your automation projects, and it provides Red Hat support.

Reference architectures

This section provides some sample Ansible architectures with different combinations of Ansible automation components designed to meet different availability requirements for your department or business. These Ansible automation components can be deployed on a single system in some environments or may require more than one system as you build out your Ansible Automation Controller or Platform.

Reference architecture 1 - Ansible Automation Controller

This architecture is very easy to deploy and will require minimum computing resources to build an Ansible Automation Controller environment. The architecture is shown in Figure 3-3.

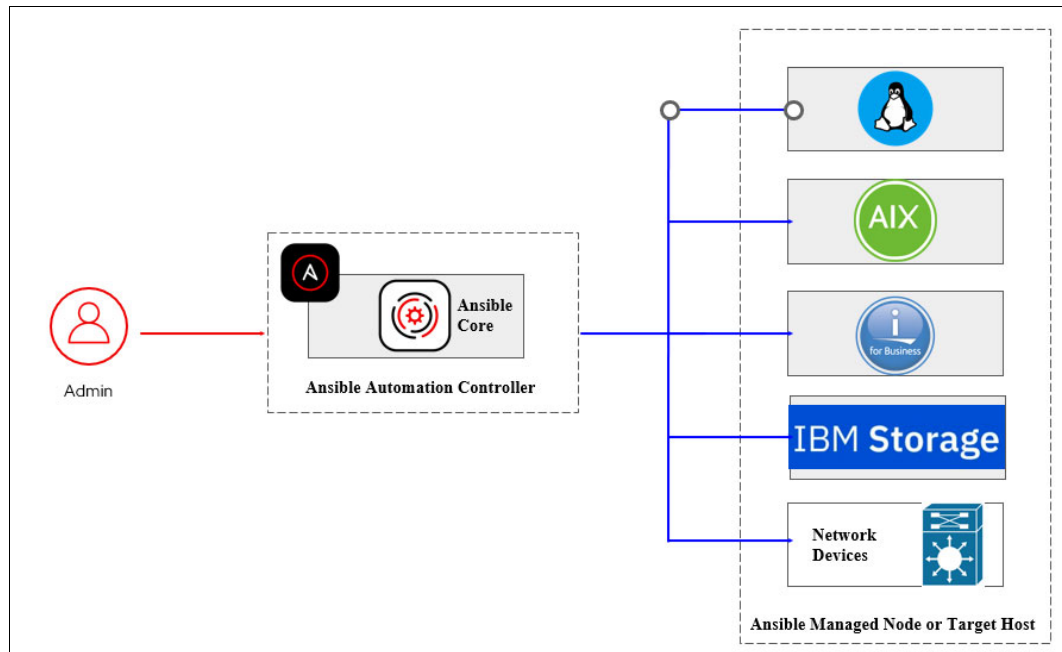


Figure 3-3 Reference architecture 1 - Ansible Automation Controller

This architecture is defined by:

- Deploy Ansible core in a single system.
- Number of nodes required:
 - One Ansible automation controller nodes
- Automation
 - Any
- Use cases:
 - Testing and development environment

Consideration: Completely command-line interface (CLI) driven, audit logs, access controls are dependent on third party software integration, no high availability. complicated day2 operation. No additional subscription or license required. Nearly any UNIX-like machine with Python 3.9 or newer is supported as the installation target. This includes Red Hat Enterprise Linux, IBM AIX, Debian, Ubuntu, macOS, BSDs, and Windows under a Windows Subsystem for Linux (WSL) distribution.

This would be a good choice for a small starter system for test and development or for a small environment with non-critical automation requirements. Support is community based.

Reference architecture 2 - Automation Platform, All-In-One without Automation Hub

This architecture is not very hard to deploy and could be started with minimum computing resources to build an Ansible Automation Platform environment. This is shown in Figure 3-4.

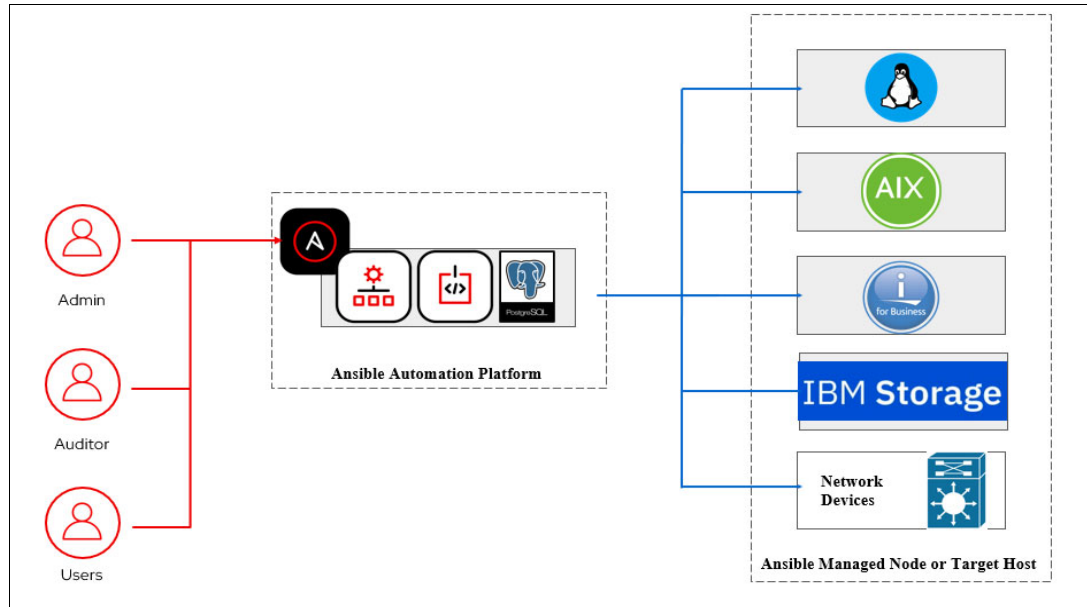


Figure 3-4 Reference architecture 2 - All-In-One without Automation Hub

This architecture is defined by:

- Deploy Red Hat Ansible Automation Platform in a single system.
- Number of nodes required:
 - One all-in-one automation controller node
- Automation
 - Any
- Use cases:
 - Testing and development environment

Consideration: Provides a graphical user interface (GUI), audit logs, and access control in build feature. Also provides a user friendly Day 2 operation automation platform. There are subscription or licenses required. This version does not provide high availability and does not use Automation Hub to manage execution images or Ansible Content Collections management. The supported deployment environment is limited to RHEL operating systems on X86 or Power systems.

This would be a good choice for a small starter system for test and development or for a small environment with non-critical automation requirements but also provides a base for expansion as you grow. Support is provided by Red Hat.

Reference architecture 3 - All-In-One with Automation Hub

This architecture builds on Reference architecture 2 - Automation Platform, All-In-One without Automation Hub by adding the Automation Hub. This is shown in Figure 3-5.

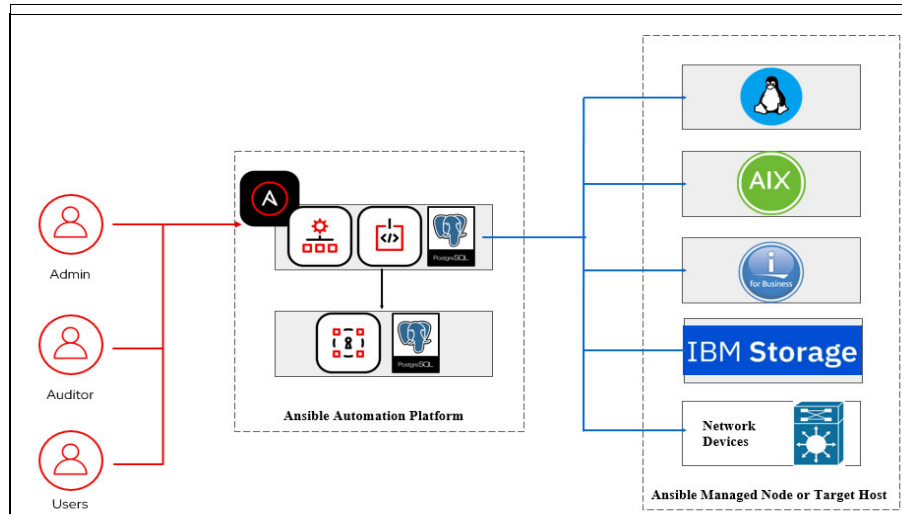


Figure 3-5 Reference architecture 3 - All-In-One with Automation Hub

This architecture is defined by:

- Deploy Red Hat Ansible Automation Platform in a single system as well as Automation Hub.
- Number of nodes required:
 - one all-in-one automation controller node.
 - one all-in-one Automation Hub node.
- Automation
 - Any
- Use cases:
 - Testing and development environment

Consideration: Enhancement of Reference architecture 2 - Automation Platform, All-In-One without Automation Hub by adding All-In-One Automation Hub. The Automation Hub adds the ability to manage your Ansible playbooks and collections which can help you automate new projects faster and more reliably. There is still no high availability and service redundancy hasn't been considered.

This would be a good choice for a small starter system for test and development or for a small production environment with non-critical automation requirements. This is has the basics of a larger scaled up solution to support growth in your environment. Support is provided by Red Hat.

Reference architecture 4 - External database server without high-availability (HA)

This architecture separates the database system to an external database provider. This is shown in Figure 3-6.

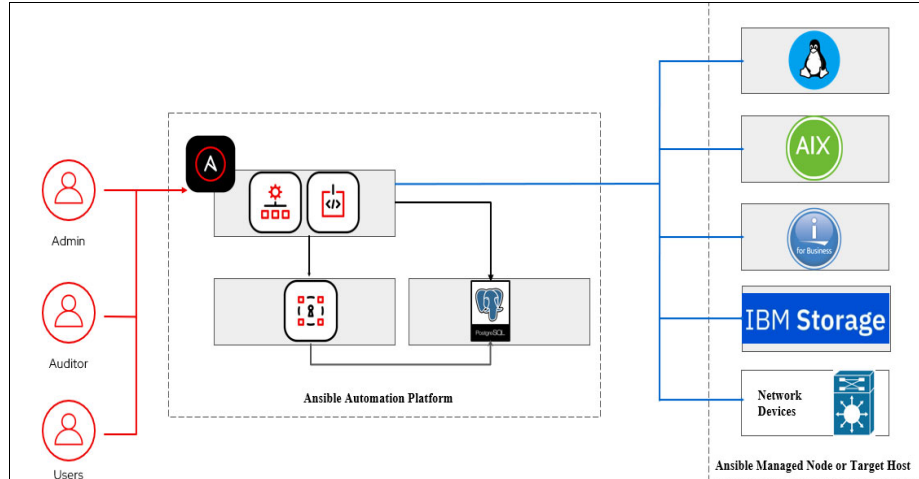


Figure 3-6 Reference architecture 4 - External database server without high-availability

This architecture is defined by:

- Deploy Red Hat Ansible Automation Platform in a single system as well as automation Hub. Separate the database system as an external database provider.
- Number of nodes required:
 - One automation controller node
 - One automation Hub node
 - One shared database node
- Automation
 - Any
- Use cases:
 - Testing and development environment

Consideration: Enhancement of Reference architecture 3 - All-In-One with Automation Hub by adding separate external databases system as well as separate system access control. There is still no high availability and service redundancy hasn't been considered.

This would be a good choice for a medium to large system for test and development or for a medium sized environment with non-critical automation requirements. Support is provided by Red Hat. This environment will scale well in terms of number of systems managed and number of projects being managed, however it does not provide any way to recover the automation platform if the controller nodes fail.

Reference architecture 5 - External database server with high-availability (HA) except Database server

This architecture builds on Reference architecture 4 - External database server without high-availability (HA) by adding high availability for the Ansible controller portions. This is shown in Figure 3-7.

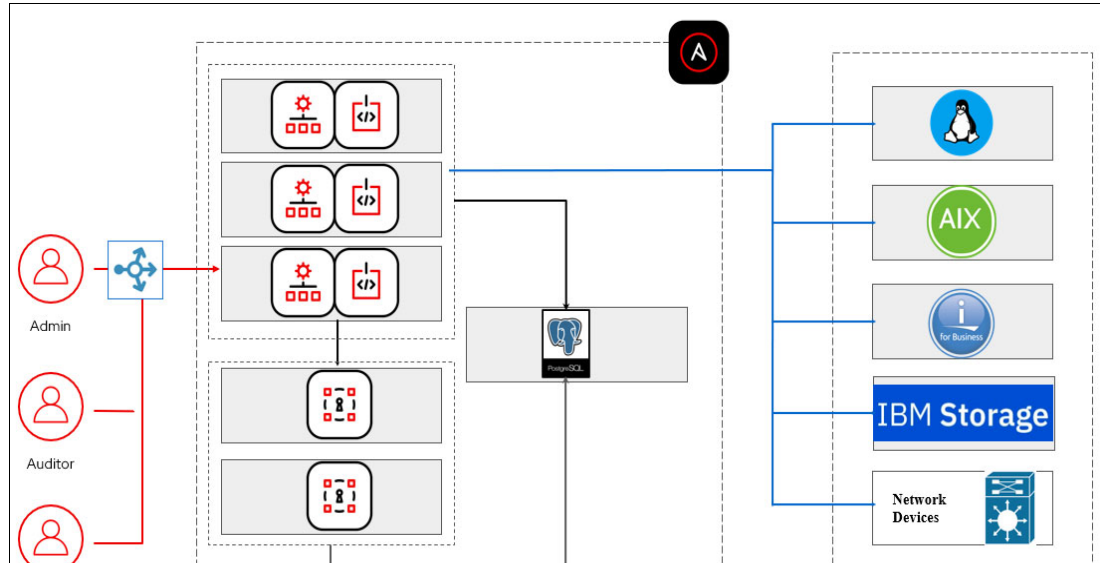


Figure 3-7 Reference architecture 5 - External database server with high-availability (HA) except database server

This architecture is defined by:

- Deploy Red Hat Ansible Automation Platform and Automation Hub in multiple systems. Separate the database system as an external database provider without high-availability (HA).
- Number of nodes required:
 - Three automation controller nodes.
 - Two automation Hub nodes.
 - One shared database node.
- Automation
 - Any
- Use cases:
 - Production environment

Consideration: Enhancement of Reference architecture 4 - External database server without high-availability (HA) by adding high-availability (HA) except database server high-availability (HA). The database server could recover from backup if needed. but RPO and RTO will be higher.

This is aimed at medium to large environments that have critical requirements for their automation environment as there are redundant nodes for the automation controller and automation Hub functions.

Reference architecture 6 - with External database Server with full high-availability (HA)

This architecture adds high availability to Reference architecture 5 - External database server with high-availability (HA) except Database server. It is shown in Figure 3-8.

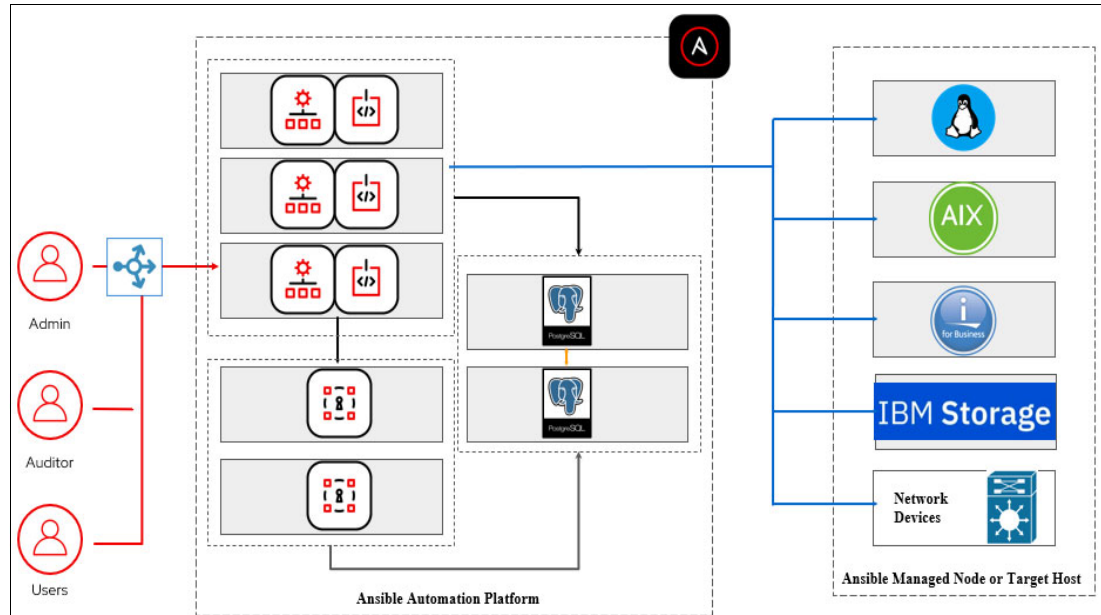


Figure 3-8 Reference architecture 6 - External database server with full high-availability

This architecture is defined by:

- Deploy Red Hat Ansible Automation Platform and Automation Hub in multiple systems. Separate the database system as an external database provider with high-availability (HA).
- Number of nodes required:
 - Three automation controller nodes
 - Two automation Hub nodes
 - Two shared database nodes
- Automation
 - Any
- Use cases:
 - Production environment

Consideration: Enhancement of Reference architecture 5 - External database server with high-availability (HA) except Database server by adding high-availability (HA) for all the components. The database server could recover from backup if needed. Minimize RPO and RTO will be higher. Automation platform is considered as the important service. But allowing access between automation controller nodes and managed nodes within the same data center with different network zones or segmentation might be complicated.

This is a great option for a highly available and highly scalable automation environment within a single site. It only lacks the ability for quick recovery to a second site in case of a site failure.

Reference architecture 7 - External database Server with full high-availability (HA) and separate execution node

This architecture adds separate execution zones to Reference architecture 6 - with External database Server with full high-availability (HA). This is shown in Figure 3-9.

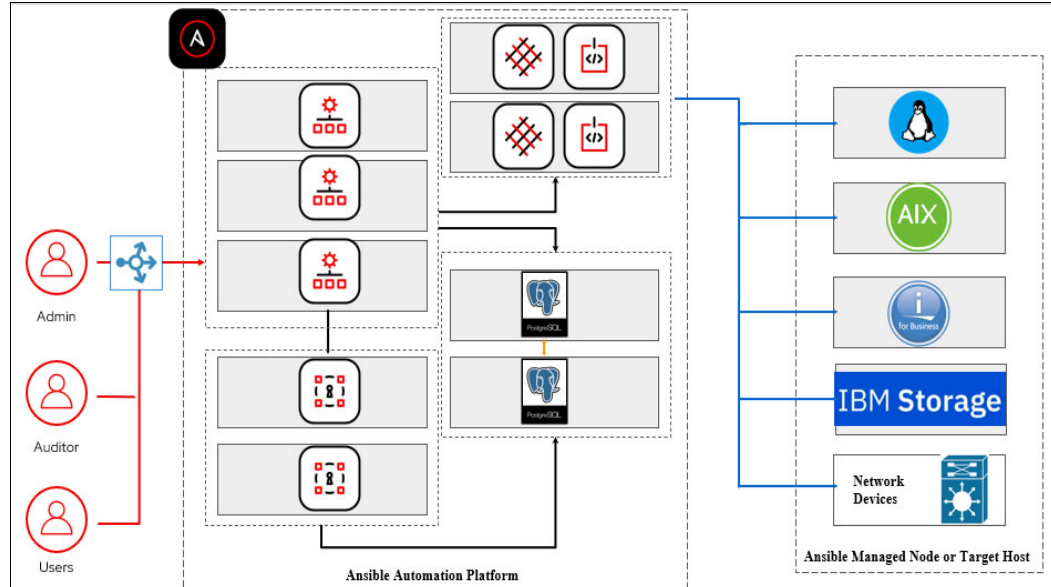


Figure 3-9 Reference architecture 7 - External database Server with full high-availability (HA) and separate execution node

This architecture is defined by:

- Deploy Red Hat Ansible Automation Platform and Automation Hub in multiple systems. Separate the database system as an external database provider with high-availability (HA). Deploy separate execution environments in different network zones.
- Number of nodes required:
 - Three automation controller nodes
 - Two execution nodes
 - Two automation Hub nodes
 - Two shared database nodes
- Automation
 - Any
- Use cases:
 - Production environment

Consideration: Enhancement of Reference architecture 6 - with External database Server with full high-availability (HA) by adding separate execution environments in different network zones. Will minimize firewall rules changes to allow access between automation controller nodes and managed nodes within the same data center with different network zones or segmentation. Site resilience consideration is still missing.

This provides a fully scalable and highly available automation solution and includes additional separation of automation activities for security. It includes redundant components but still does not address disaster recovery for site failure.

Reference architecture 8 - External database server with full high-availability (HA) and disaster recovery (DR) - Independent Operation

This architecture adds a disaster recovery environment. Automation can be managed from either the primary location or the disaster recovery location. This is shown in Figure 3-10.

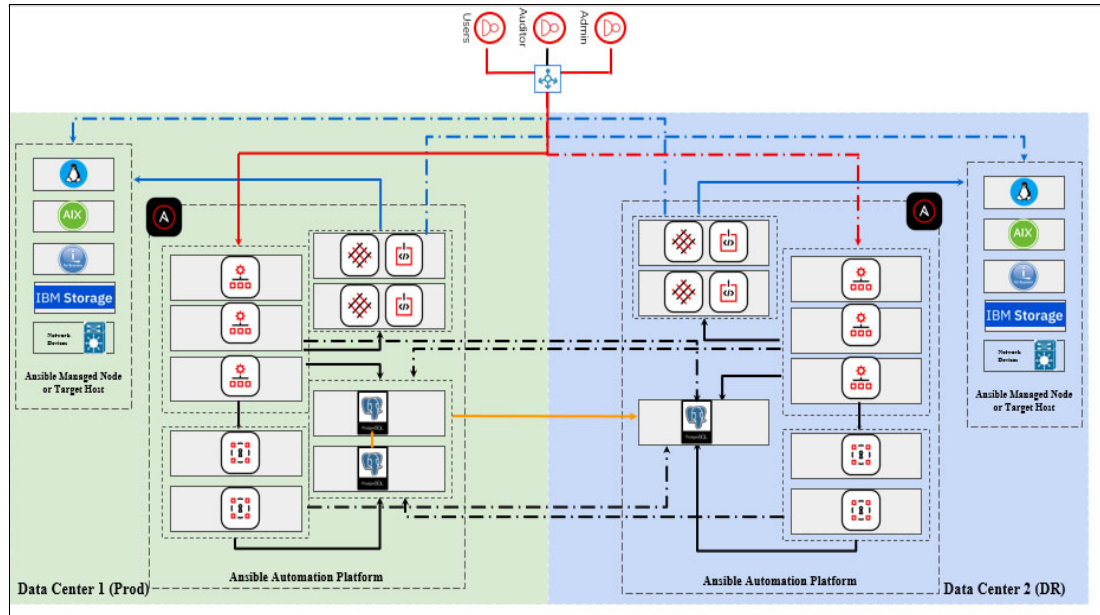


Figure 3-10 Reference architecture 8 - External database server with full high-availability (HA) and disaster recovery (DR) - Independent Operation

This architecture is defined by:

- Deploy Red Hat Ansible Automation Platform and Automation Hub in multiple systems. Separate the database system as an external database provider with high-availability (HA). Deploy separate execution environments in different network zones.
- Number of nodes required:
 - Three automation controller nodes per site
 - Two execution nodes per site
 - Two automation Hub nodes per site
 - Two shared database nodes Prod site
 - One shared database nodes DR site
- Automation
 - Any
- Use cases:
 - Production environment
 - Disaster recovery environment

Consideration: Enhancement of Reference architecture 7 - External database Server with full high-availability (HA) and separate execution node by adding a separate disaster recovery environment. At-least one Automation platform from either site can perform the Automation operation on all the managed nodes across the site. That is, The Automation operation on all the managed nodes across the site will not be impacted, if one of the Automation platforms is down. Huge firewall rules changes might be required to allow access between automation controller nodes and managed nodes across the sites.

Fully scalable and redundant solution with disaster recovery consideration.

Reference architecture 9 - External database server with full high-availability (HA) and disaster recovery (DR) - Joint Operation

This architecture adds joint operation to what is provided in Reference architecture 8 - External database server with full high-availability (HA) and disaster recovery (DR) - Independent Operation. This is shown in Figure 3-11.

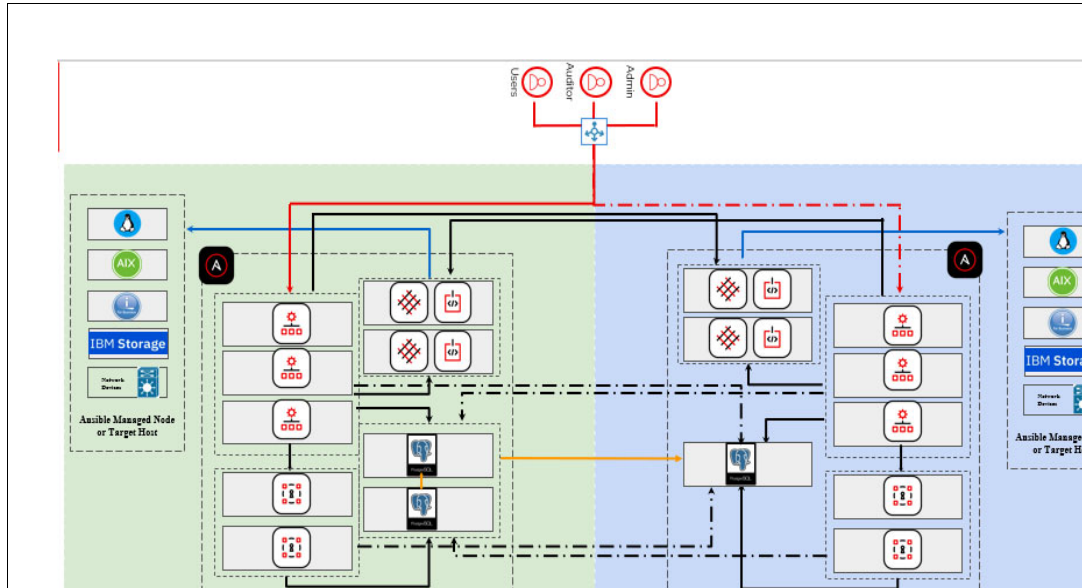


Figure 3-11 Reference architecture 9 - External database server with full high-availability (HA) and disaster recovery (DR) - Joint Operation

This architecture is defined by:

- Deploy Red Hat Ansible Automation Platform and Automation Hub in multiple systems. Separate the database system as an external database provider with high-availability (HA). Deploy separate execution environments in different network zones.
- Number of nodes required:
 - Three automation controller nodes per site
 - Two execution nodes per site
 - Two automation Hub nodes per site
 - Two shared database nodes Prod site
 - One shared database nodes DR site
- Automation
 - Any
- Use cases:
 - Production environment
 - Disaster recovery environment

Consideration: Enhancement of Reference architecture 8 - External database server with full high-availability (HA) and disaster recovery (DR) - Independent Operation by adding a separate disaster recovery environment. One Automation platform from either site can perform the Automation operation on all the managed nodes across the site, through the execution node in the respective site. That is, The Automation operation on all the managed nodes across the site will not be impacted, if one of the Automation platforms is partially down, except the execution nodes. Will minimize firewall rules changes to allow access between automation controller nodes and managed nodes across the site. That is the firewall rules changes required only for the access between automation controller nodes and execution nodes across the site only. Other required access will be maintained within the site only.

Table 3-1 helps to differentiate the implementation considerations between Reference architecture 8 - External database server with full high-availability (HA) and disaster recovery (DR) - Independent Operation and Reference architecture 9 - External database server with full high-availability (HA) and disaster recovery (DR) - Joint Operation.

Table 3-1 Differentiation between reference architecture 8 and referee architecture 9

Characteristic	Reference architecture 8	Reference architecture 9
Operation Dependency	Totally independent and tolerant of any of the full sites down.	Partially independent and tolerant of any of the partial sites down. That is, Execution nodes have to be accessible for automation activities across the sites.
Execution Node	The control plane (nodes) is connected with execution nodes on the same side only.	The control plane (nodes) is connected with all the execution nodes across the sides.
Firewall Rules Changes	In a large automation environment, a large number of firewall rules needs to be changed.	In a large automation environment, the very minimum number of firewall rules needs to be changed.
Network Bandwidth Utilization	In a large automation environment, the network bandwidth utilization will be comparatively high across the sides.	In a large automation environment, the network bandwidth utilization will be minimum across the sides.
Network Latency	Not suitable for high network latency across the sides. That is, the network latency between the sides must be very negligible. so that the execution nodes with automation execution environments and the target host or endpoints could be placed in two different locations, and enable automation for edge use cases.	Suitable for high network latency across the sides. Because the execution plane (nodes) only runs user-space jobs, they may be geographically separated, with high latency, from the control plane (nodes). That is, the execution nodes with automation execution environments are placed in different locations that are closer to the target host or endpoints to reduce latency and enable automation for edge use cases.

Conclusion

Looking at the above reference architectures will give you some ideas on how to design your automation environment. You can create your supported architecture with the help of the reference architectures provided here, choosing the components that meet your requirements best. However, there are some further things to consider for when adding and integrating additional solution components.

Consider the following points as you design your environment:

▶ **Database Nodes**

Database high-availability (HA) clusters can be configured with RHEL native HA cluster solution, called Pacemaker, or can use PostgreSQL HA solutions known as Primary-Standby and Primary-Primary architectures.

▶ **Automation Controller Nodes**

At Least two nodes can be considered for the Automation Controller nodes high-availability (HA).

▶ **Automation Hub Nodes**

Automation Hub could be optional and alternative solutions can be used. Can be reduced to one node in the disaster recovery (DR) site in case of any resource limitation.

▶ **Third-party Services Integration**

It is highly recommended to integrate and configure with third-party services as per requirement, such as:

- Source code management:

Manage project and playbooks through source code management system, including Git, Subversion, and Mercuria.

- Notification methods:

Use Email, Grafana, Slack or similar tools.

- Authentication:

LDAP, SAML, token-based authentication.

- Logging:

Consider logging aggregation services for monitoring and data analysis of your systems, including Splunk, Loggly, Sumologic, Elastic stack (formerly ELK stack).

3.1.3 Enterprise-ready environment

As your organization progresses in Ansible and automation culture adoption, you may want to take the next step. A real enterprise consists more than of one team. It has many development and operations teams and it has many environments. It also has many different non-functional requirements for automation, such as the requirement for centrally managed authentication through Active Directory or requirements for delivering security-related logs to the organization's SIEM (Security Information Event Monitoring) system. Many of these requirements are already integrated in Red Hat Ansible Automation Platform and can be implemented using those features. However, some of them will require third party software.

In an enterprise environment, all source code must be saved in some source control repository. It enables you to track changes to the code and see who did what. It also enables you to separate projects and teams. The same concepts apply to automation source code – your playbooks and roles. Your Windows administration team has nothing to do with AIX or IBM i automation. On the other hand AIX operations usually have little interference with Microsoft SQL Server or Sharepoint resources managed by other teams.

The some of the most common control management tools used in enterprises are Github Enterprise and Gitlab Enterprise. They are based on open source *git* project and you can use *git* on your Linux, AIX or IBM i server to work with them.

After a change to a source code is committed, the new code must be tested. This applies to automation even more, because if someone made a small mistake in the automation code, it can cause problems across your whole application deployment or infrastructure. The testing can be as simple as doing a syntax check or can involve more complex integration testing where the whole infrastructure is built and the application is deployed into a special testing environment. Source control management tools like Github Enterprise and Gitlab Enterprise have their own set of continuous integration (CI) tools, but if you wish to use other tools to manage the source code, you may also use the open source tool Jenkins to build your integration pipeline.

Another very important part of the process is to check that the source repository does not contain any passwords, tokens or other secrets. Your secrets must be stored in Ansible Automation Platform or in some other vault tool like Hashicorp Vault, but not in the source code. Modern source code management tools like Github and Gitlab can integrate with all common security tools to automate source code scanning.

When the whole testing process is completed successfully, the code can be deployed into your production infrastructure. It can be done automatically, using continuous delivery, if you choose. When using Ansible Automation Platform, we usually get a new version of a project after synchronizing it. Just as a side note you may want to automate Ansible Automation Platform the same way you automate your other applications.

In an enterprise environment, you want your automation to be predictable, which is possible if your code is tested before it goes to production. It is also important to ensure that the code you rely upon is stable – this includes every role, every module and every collection. To meet this goal you can use the Red Hat curated Automation Hub instead of Ansible Galaxy as the source of your collections. Another option is to use Private Automation Hub, which is provided by Ansible Automation Platform. Private Automation Hub allows you to upload (or synchronize) only the content you really need for your automation code, this allows you or your IT security team to validate and approve the components that are used in your organization.

In the simplest form of deployment we used just one Ansible Controller to manage all nodes. In the enterprise-grade deployment you install Ansible Automation Controllers according to your infrastructure requirements. You may have separate Controllers for each stage (development, test, production) or you may install them based on your network configuration - separate Controllers for office network, for “normal” servers, for high priority servers, for DMZ servers, and so on. It makes your architecture more complex but it is easier to control which projects access which resources.

Another component of Ansible Automation Platform which is specifically designed for enterprise environments, is Event-Driven Automation. Consider the following questions:

- What happens in the environment if someone provisions a server without using your automation?
- What happens if a new user must be created on several servers?
- What happens if a filesystem on a server needs more space?

These are all use cases for Event-Driven Automation (EDA).

With Event-Driven Automation, you develop playbooks for each use case – to configure a new server, to create a user on a server, to expand a filesystem –and then connect your external systems like PowerVC (in case of provisioning), ticketing system (in case of new user creation) or monitoring (in case of filesystem) to EDA. Within EDA you define the policies and rules and EDA will run the appropriate playbooks when a defined event happens, so you can build a fully automated enterprise. More use-cases for EDA can be found in section 1.3.3, “Event-Driven automation” on page 14.

One more aspect of a complex automation architecture is organizational in nature. As you have multiple components to manage – several Ansible Automation Controllers, Private Automation Hub, Event-Driven Automation and others – the job of managing the environment can not be a side job. In this case your organization needs a separate automation team which automates and manages Ansible Automation Platform. This architectural pattern applies only to enterprises with a large infrastructure and a large number of applications.

While It is not easy to make a graphic representation of such a complex environment, Figure 3-12 provides an example of what is possible. Your environment may be even more complex and you may need to spend some time defining and building your automation architecture.

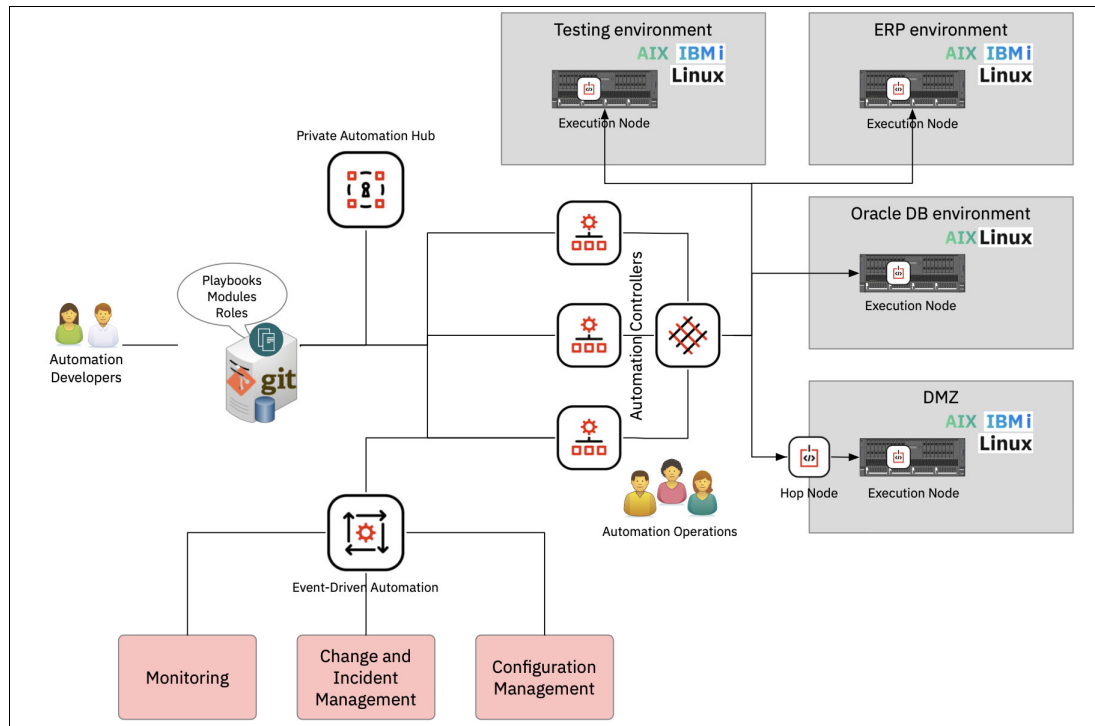


Figure 3-12 Sample Ansible Automation Platform implementation

3.1.4 Develop an “automation first” attitude

The choice of how to get started with Ansible depends on your team, your budget, and your time line. You could start with the simplest form of deployment and later grow into a full-pledged Ansible Automation Platform installation as you integrate automation across your enterprise (including third party applications). However, if you are ready to design an automation architecture for your whole enterprise from the beginning, you can start with Ansible Automation Platform and employ all the wonderful features it brings.

In any case, don't look at your architecture as something permanent. Automation evolves and automation practices evolve as well. Your environment is live, be prepared to live with it and enhance it every time you require a new feature or a new integration, and be prepared to eliminate unnecessary or unused features.

However, the most important decision is not the software components you use. The most important step is to create an “automation first” environment. Before doing any task on your systems, take a step back, take your time and think – can I automate this function using Ansible? The obvious answer is often, yes. Then automate the task and let the job to be done

by Ansible. When you get to this point you no longer need *root* or QSECOFR privileges on your systems. All you need is that your systems are connected to your automation platform and you can execute Ansible playbooks there.

3.2 Choosing the Ansible controller node

As noted in our previous discussions, Ansible can be installed on nearly any system. In order to choose the best location and system to run your Ansible control node requires that you understand your environment and your automation requirements.

Before choosing the right controller node for Ansible, you must answer a simple question: Which systems do you plan to manage with Ansible? Consider the following cases:

- ▶ If you only want to manage your IBM i database, the answer is very easy. Use your IBM i as the Ansible controller.
- ▶ If you want to manage your AIX environment, you may want to install Ansible on your network installation manager (NIM) server. NIM is usually the central point of AIX infrastructure and already has access to all AIX servers and many times, NIM already uses open SSH connections between the NIM server and the NIM clients.
- ▶ If you have SAP HANA on IBM Power, or other Linux applications on IBM Power, you may want to use an existing Linux on Power LPAR to install Ansible. This choice has one big advantage. Ansible is developed under Linux and with Linux first in mind. As of the time of writing this Redbook you can install Ansible Core 2.15 on Linux on Power, but Ansible Core 2.14 is the latest version available on AIX. While most modules and collections support Ansible 2.9 or later, if you have something specific that requires a newer Ansible version, your only choice is Linux.
- ▶ Of course you may use Linux on x86 for Ansible controller node. This is usually an obvious choice if your environment already has Ansible on an x86 Linux server. Then you don't need to install anything, you can use the existing server.

No matter which Ansible controller node you choose, it must have SSH access to all of the systems you want to manage. Check that it has such access or ensure that the connection can be made through firewalls and security zones.

It probably doesn't make sense to install your Ansible controller node in AWS to manage AIX servers on-premise or in IBM Cloud PowerVS. You may want to have your Ansible controller node as close to the managed servers as possible.

If you place your Ansible controller node in the DMZ together with other servers, it will simplify the connection between the Ansible controller and the target hosts, but it also might be difficult to upload your playbooks and roles to it. It might be easier to install Ansible on a server outside of the DMZ and use some jump host for playbook execution. This is probably a topic of discussion in a meeting with your network and IT security teams.

Now that you have chosen which server to use for your Ansible controller node and have consulted with your IT security and network teams and you have access from your Ansible controller node to all your systems by SSH, you are ready to install Ansible!

3.3 Installing your Ansible control node

This section describes how to install the Ansible code on your IBM Power based controller node. Ansible is an excellent tool for configuration management, automated deployment and

orchestration. There are two components to consider when using Ansible, the first is the control node which executes the playbooks and manages the automation, and the second is the managed node or client which is the device being automated (often called the target machine or device). As we have discussed, Ansible is agentless which means that it can communicate with machines or devices without requiring that an application or service be installed on that managed node. This is one of the main differences between Ansible and other similar applications like Puppet, Chef, CFEngine and Salt.

The Ansible controller is often called Ansible Engine (old name) or Ansible Core (new name). Ansible Core provides a command-line interface (CLI) to manage your Ansible automation environment. For some administrators, the CLI based approach is intimidating and they are looking for a graphical user interface (GUI) instead. For GUI based management, you can choose to use the Red Hat Ansible Automation Platform which provides the GUI interface to Ansible Core and also provides additional management capabilities.

Ansible Core is available and supported on all operating systems supported by IBM Power – AIX, IBM i, and Linux on Power. Red Hat Ansible Automation Platform is also available for IBM Power environments, but it is only supported by Linux on Power. Currently, the Linux on Power support for Ansible Automation Platform is in Technical Preview, full support is expected to be announced in the near future.

3.3.1 Linux as an Ansible controller

Ansible Installation on RHEL

Ansible Engine and Ansible Core cannot be installed simultaneously on a RHEL 8 system and the installer (RPMs) will be located in two different rpm repositories. Ansible Core is included in the RHEL 8.6 and later as well as the RHEL 9 operating system version, under the AppStream repository, and can be installed by running the rpm or dnf (newer version of yum) command, For more details see: <https://access.redhat.com/articles/6393361>.

To install Ansible on Red Hat Enterprise Linux follow these steps:

1. Verify the system identity, name, organization name and organization ID provided upon subscription registration as seen in Example 3-1.

Example 3-1 Verify subscription-manager details

```
# subscription-manager identity
system identity: 8e73cf0a-4651-4b0d-95c1-b0b73a886785
name: app24allinone.example.com
org name: 11009103
org ID: 11009103
```

2. Verify the current status of the products and attached subscriptions for the system as shown in Example 3-2.

Example 3-2 Verify subscription-manager status

```
# subscription-manager status
+-----+
  System Status Details
+-----+
Overall Status: Disabled
Content Access Mode is set to Simple Content Access. This host has access to content,
regardless of subscription status.
System Purpose Status: Disabled
```

3. Verify the RPM repository is configured and enabled using the command shown in Example 3-3.

Example 3-3 Verify RPM repository configuration

```
# yum repolist
Updating Subscription Management repositories.
repo id                                repo name
rhel-8-for-ppc64le-appstream-rpms      Red Hat Enterprise Linux 8 for ppc64le
- AppStream (RPMs)
rhel-8-for-ppc64le-baseos-rpms        Red Hat Enterprise Linux 8 for ppc64le
- BaseOS (RPMs)
```

4. Install the ansible-core rpm in the system using the following command:

```
# dnf install ansible-core python3-virtualenv vim
```

Note: Make sure your system is connected to the right rpm repository. If the system directly connects with the internet, then make sure the subscription is configured and enable the correct repository. For more information about using subscription manager refer to <https://access.redhat.com/solutions/253273>.

Verify the Installation of Ansible on RHEL:

Once the Ansible Core rpm installation completes in the system, it will have the configuration file and binaries that are commonly used and shown in Table 3-2.

Table 3-2 List of files that associated with ansible-core rpm

Important Files and Executables	Description
/etc/ansible/ansible.cfg	Default configuration file that comes with RPM packages and it will have all the required settings, the location of module search path, module, executable, inventory file and so on.
/etc/ansible/hosts	Sample inventory for the managed node or target host where automation tasks will be executed.
/usr/bin/ansible-config	<p>Command to view the effective Ansible configuration details and the file location. Which configuration file is used depends on the file location. The order of importance is:</p> <ul style="list-style-type: none"> ▶ /etc/ansible/ansible.cfg - Default configuration file, used if present ▶ ~/.ansible.cfg - User configuration file, overrides the default config if present ▶ ./ansible.cfg - Local configuration file that is located in the current working directory assumed to be 'project specific' and overrides the rest if present. ▶ ANSIBLE_CONFIG - Specify override location for the Ansible config file <p>For example the command below can create a sample configuration for you.</p> <pre># ansible-config init --disabled -t all > ansible.cfg</pre>

Important Files and Executables	Description
/usr/bin/ansible	Command to define and run a single task 'playbook' against a set of hosts. For example: Run the shell module to execute a command # ansible all -i hosts -m shell -a "hostname"
/usr/bin/ansible-console	Command that allows for on-the-fly execution of Ansible modules or arbitrary commands to the hosts. For example: # ansible-console -i hosts --limit all -u root
/usr/bin/ansible-doc	Command to display information on specific modules installed in Ansible libraries. For example: To check the <i>copy</i> module documentation. # ansible-doc copy
/usr/bin/ansible-playbook	Command to run Ansible playbooks, executing the defined tasks on the targeted hosts. For example: To run a sample playbook. # ansible-playbook myplaybook.yml
/usr/bin/ansible-vault	Command that can be used as an encryption/decryption utility for Ansible that can encrypt any structured data file used by Ansible.

To continue with the verification and configuration do the following:

1. Generate the configuration file using the command below:

```
# ansible-config init --disabled -t all > ansible.cfg
```

2. Display the effective configuration and the its configuration file location, in terms of current working location or directory, as shown in Example 3-4.

Example 3-4 Display effective configuration

```
# ansible-config --version
ansible-config [core 2.14.2]
  config file = /root/ansible.cfg
  configured module search path = ['/root/.ansible/plugins/modules',
'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3.11/site-packages/ansible
  ansible collection location =
/root/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible-config
  python version = 3.11.2 (main, Jun 6 2023, 07:39:01) [GCC 8.5.0 20210514 (Red Hat
8.5.0-18)] (/usr/bin/python3.11)
  jinja version = 3.1.2
  libyaml = True
```

3. Verify the inventory file name and location as shown in Example 3-5.

Example 3-5 Verify inventory file location

```
# grep -vE "^#|^;" /root/ansible.cfg|grep -v ^$
[defaults]
inventory=./hosts
[privilege_escalation]
[persistent_connection]
[connection]
[colors]
[selinux]
[diff]
[galaxy]
[inventory]
[netconf_connection]
[paramiko_connection]
[jinja2]
[tags]
[runas_become_plugin]
[su_become_plugin]
[sudo_become_plugin]
[callback_tree]
[ssh_connection]
[winrm]
[inventory_plugins]
[inventory_plugin_script]
[inventory_plugin_yaml]
[url_lookup]
[powershell]
[vars_host_group_vars]
```

4. Verify the inventory file configuration as shown in Example 3-6.

Example 3-6 Verify inventory configuration

```
# cat /etc/ansible/hosts
192.168.121.203
```

5. Run an ad-hoc command to verify the functionality as shown in Example 3-7.

Example 3-7 Test Ansible functionality with ad-hoc command

```
# ansible all -i hosts -m shell -a "hostname" -u root -k
SSH password:
192.168.121.203 | CHANGED | rc=0 >>
localhost.localdomain
```

Additional Preparation and Configuration for Ansible on RHEL

A recommendation is to create a separate user to manage automation activities, generate ssh keys for managed nodes access, create a virtual environment for specific python versions, set environment variables for better working environment and so on. The following steps are recommended to make your Ansible experience better:

- ▶ Create a user called *ansible* using the following command:

```
# useradd -m -c "Ansible Controller User" ansible
```

- ▶ Install any additional python libraries or modules depending on requirements. Use the following command:

```
# dnf install python3-pyOpenSSL python3-winrm python3-netaddr python3-psutil
python3-setuptools
```

Note: If any specific python libraries or modules are not available or not shipped with RHEL OS in rpm format, they can be installed via **pip** (the python packages manager). You can create a virtual environment, like a virtual machine or Linux *chroot*, that will have an isolated structure of lightweight directories separated from the actual operating system python directories to allow you to use different versions of python modules, files, or configurations.

- ▶ Create a `~/.vimrc` file to customize the vim editor configuration to use the 2 space indentation for yaml file editing as shown in Example 3-8.

Example 3-8 Modify vim editor configuration

```
# cat << 'EOF' >> ~/.vimrc
autocmd FileType yaml setlocal ts=2 sts=2 sw=2 expandtab
autocmd FileType yml setlocal ts=2 sts=2 sw=2 expandtab
EOF
```

- ▶ Generate and copy the ssh key from Ansible Automation Controller node to managed nodes using the following commands:

```
# ssh-keygen
# ssh-copy-id root@192.168.121.203
```

Ansible Automation Platform Installation on RHEL

The Ansible Automation Platform can be installed and configured in IBM Power Systems. The step-by-step installation is shown here:

1. Download the Red Hat Ansible Automation Platform installer from the [Red Hat product download site](#). From the Red Hat product download site, select the product called “Red Hat Ansible Automation Platform” as shown in Figure 3-13.

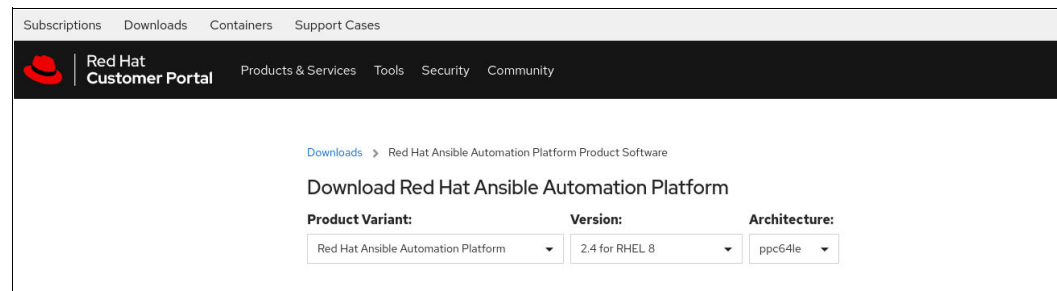


Figure 3-13 Select version and architecture for Ansible Automation Platform package download

The download list will be available once you select the Red Hat Ansible Automation Platform version and the architecture from the product software download page.

The software download page is shown in Figure 3-14.

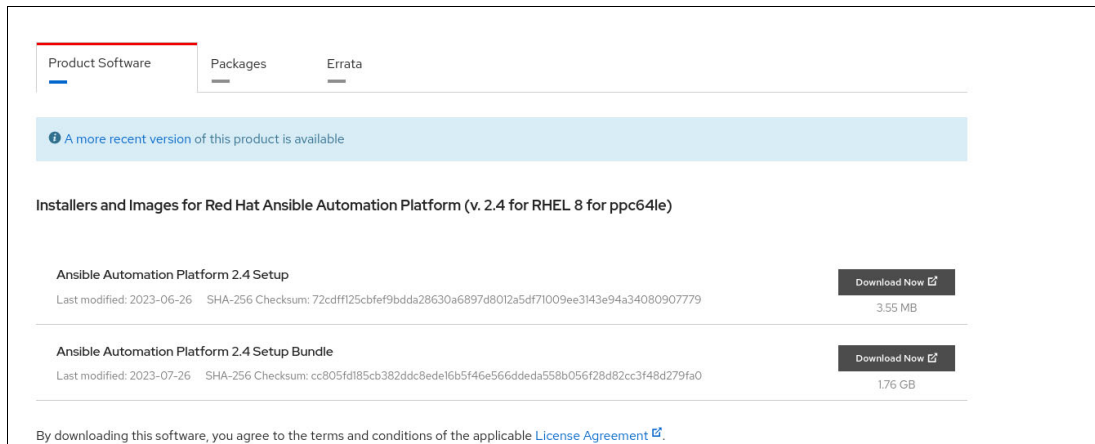


Figure 3-14 List of Ansible Automation Platform package bundles that can be downloaded

Now download the bundle package. An example file name could be:

ansible-automation-platform-setup-bundle-2.4-1.2-ppc64le.tar.gz

2. Copy that tar file in the system and extract the files as shown in Example 3-9.

Example 3-9 Copy file and extract

```
# tar xvfz ansible-automation-platform-setup-bundle-2.4-1.2-ppc64le.tar.gz
# ls -l
:::~::~:Some Output Removed::~:~::~:~::~:
drwxrwxrwx. 5 root root      4096 Jul 23 10:58 ansible-automation-platform-setup-bundle-2.4-1.2-ppc64le
-rw-r--r--. 1 root root 2068376768 Jul 23 00:06 ansible-automation-platform-setup-bundle-2.4-1.2-ppc64le.tar.gz
:::~::~:Some Output Removed::~:~::~:~::~:
```

3. Go to the extracted directory and configure the inventory file with a vim editor for the all-in-one installation scenario. This process is shown in Example 3-10.

Example 3-10 Configure inventory file

```
# cd ansible-automation-platform-setup-bundle-2.4-1.2-ppc64le/
# ls -l
:::~::~:Some Output Removed::~:~::~:~::~:

-rw-rw-rw-. 1 root root    530 Jun 26 19:55 README.md
drwxrwxrwx. 5 root root   4096 Jun 26 19:43 bundle
drwxrwxrwx. 3 root root   4096 Jun 26 19:38 collections
drwxrwxrwx. 2 root root   4096 Jun 26 19:38 group_vars
-rw-rw-rw-. 1 root root   8653 Jul 23 10:30 inventory
-rwxrwxrwx. 1 root root  14780 Jun 26 19:38 setup.sh

# vim inventory
# grep -v ^# inventory |grep -v ^$
[automationcontroller]
bs-rbk-lnx-1.power-iaas.cloud.ibm.com node_type=hybrid
[automationcontroller:vars]
peers=execution_nodes
[execution_nodes]
[automationhub]
[automationedacontroller]
[database]
[sso]
[all:vars]
```

```

admin_password='Redhat123'
pg_host=''
pg_port=5432
pg_database='awx'
pg_username='awx'
pg_password='Redhat123'
pg_sslmode='prefer' # set to 'verify-full' for client-side enforced SSL
registry_url='registry.redhat.io'
registry_username=''
registry_password=''
receptor_listener_port=27199
automationedacontroller_admin_password=''
automationedacontroller_pg_host=''
automationedacontroller_pg_port=5432
automationedacontroller_pg_database='automationedacontroller'
automationedacontroller_pg_username='automationedacontroller'
automationedacontroller_pg_password=''
sso_keystore_password=''
sso_console_admin_password=''

```

Note: The legacy execution environment (`ee_29_enabled=true`) is not supported for Power Systems. If the `ee_29_enabled = true` then you will receive errors as shown in Figure 3-15.

```

TASK [ansible.automation_platform_installer.preflight : Check the architecture for ee-29 container image] ***
fatal: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]: FAILED! => {
  "assertion": "ansible_architecture == 'x86_64'",
  "changed": false,
  "evaluated_to": false,
  "msg": "The ee-29 container image is only available on x86_64 architecture"
}
NO MORE HOSTS LEFT *****

```

Figure 3-15 Error caused by `ee_29_enabled = true`

- Run the Red Hat Ansible Automation Platform setup script to start the installation. This is shown in Figure 3-16.

```

oryPreserve": "no", "RuntimeMaxUsec": "infinity", "SameProcessGroup": "no", "SecureBits": "0", "SendSIGHUP": "no", "SendSIGKILL": "yes", "Slice": "system.slice",
"StandardError": "inherit", "StandardInput": "null", "StandardOutputData": "", "StandardOutput": "journal", "StartLimitAction": "none", "StartLimitBurst":
"5", "StartLimitIntervalUsec": "10s", "StartupBlockWeight": "[not set]", "StartupCPUShares": "[not set]", "StartupCPUWeight": "[not set]", "StartupIOWeight":
"[not set]", "StateChangeTimestamp": "Wed 2023-08-23 10:16:14 EDT", "StateChangeTimestampMonotonic": "1036299288", "StateDirectoryMode": "0755", "StatusErr
no": "0", "StopWhenUnneeded": "no", "SubState": "dead", "SuccessAction": "none", "SyslogFacility": "3", "SyslogLevel": "6", "SyslogLevelPrefix": "yes", "Syslo
gPriority": "30", "SystemCallErrorNumber": "0", "TTYReset": "no", "TTYVHangup": "no", "TTYVTDisallocate": "no", "TasksAccounting": "yes", "TasksCurrent": "[no
t set]", "TasksMax": "97196", "TimeoutStartUsec": "1min 30s", "TimeoutStopUsec": "1min 30s", "TimerSlackNsec": "50000", "Transient": "no", "Type": "forking",
"UID": "[not set]", "UMask": "0022", "UnitFilePreset": "disabled", "UnitFileState": "enabled", "UtmpMode": "init", "WantedBy": "automation-controller.service
multi-user.target", "WatchdogTimestampMonotonic": "0", "WatchdogUsec": "0"}}

PLAY [Post-install insights setup] *****

TASK [include_role : ansible.automation_platform_installer.misc] *****

TASK [ansible.automation_platform_installer.misc : Ensure insights-client is installed] ***
ok: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com] => {"changed": false, "msg": "Nothing to do", "rc": 0, "results": []}

TASK [ansible.automation_platform_installer.misc : Register with Insights] *****
changed: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com] => {"changed": true, "cmd": ["insights-client", "--register"], "delta": "0:01:38.333430", "end": "2023-08-23
10:18:08.268092", "msg": "", "rc": 0, "start": "2023-08-23 10:16:29.934662", "stderr": "", "stderr_lines": [], "stdout": "Successfully registered host bs-rbk-
lnx-1.power-iaas.cloud.ibm.com\nAutomatic scheduling for Insights has been enabled.\nStarting to collect Insights data for bs-rbk-lnx-1.power-iaas.cloud.ibm.c
om\nUploading Insights data.\nSuccessfully uploaded report from bs-rbk-lnx-1.power-iaas.cloud.ibm.com to account 5910538.\nView the Red Hat Insights console a
t https://console.redhat.com/insights/", "stdout_lines": ["Successfully registered host bs-rbk-lnx-1.power-iaas.cloud.ibm.com", "Automatic scheduling for Insig
hts has been enabled.", "Starting to collect Insights data for bs-rbk-lnx-1.power-iaas.cloud.ibm.com", "Uploading Insights data.", "Successfully uploaded rep
ort from bs-rbk-lnx-1.power-iaas.cloud.ibm.com to account 5910538.", "View the Red Hat Insights console at https://console.redhat.com/insights/"]}

PLAY [Post-install cleanup] *****

TASK [Remove stale packages] *****
ok: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com] => {"changed": false, "msg": "Nothing to do", "rc": 0, "results": []}

PLAY RECAP *****
bs-rbk-lnx-1.power-iaas.cloud.ibm.com : ok=380 changed=177 unreachable=0 failed=0 skipped=207 rescued=0 ignored=6
localhost : ok=0 changed=0 unreachable=0 failed=0 skipped=1 rescued=0 ignored=0

The setup process completed successfully.
Setup log saved to /var/log/tower/setup-2023-08-23-09:56:26.log.
[root@bs-rbk-lnx-1 ansible-automation-platform-setup-bundle-2.4-1.2-ppc64le]#

```

Figure 3-16 Screen shot from installation script

Note: The default minimum RAM size is 8GiB. This can be modified for non-production or a testing environment by changing the default configuration file located at:

collections/ansible_collections/ansible/automation_platform_installer/roles/pre-flight/defaults/main.yml

To adjust the minimum RAM size, modify the `required_ram` entry before continuing the installation. As an example:

```
required_ram: 4000
```

- Once installation successfully completes, login to Ansible Automation Platform User Interface as shown in Figure 3-17.

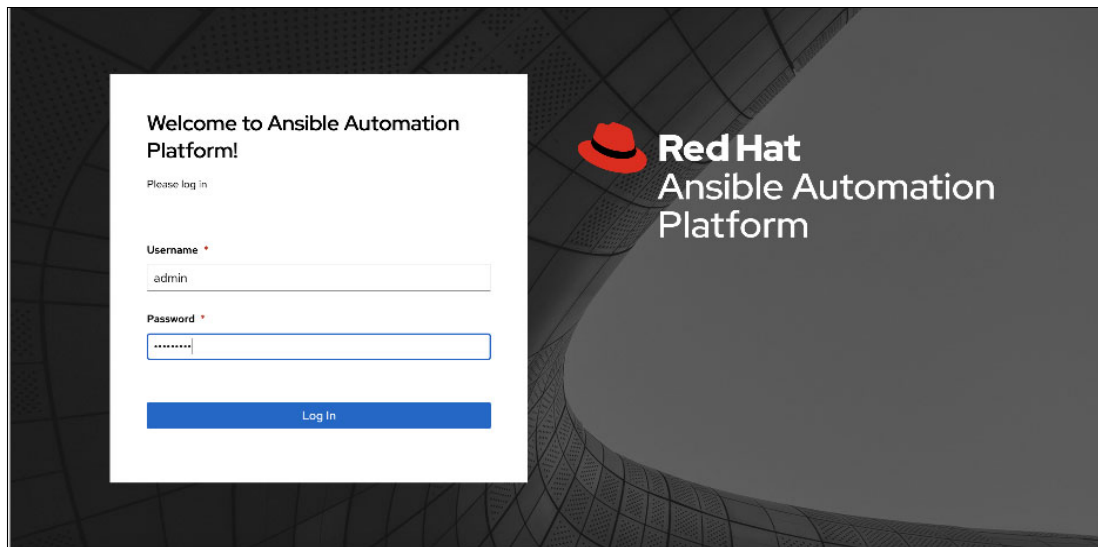


Figure 3-17 Ansible Automation Platform login page

- When you login the first time, you will need to configure the subscription manager and activate the subscription. In a disconnected or restricted environment (that is no internet access from the system), you must first create a manifest file, allocate the Red Hat software subscriptions with Ansible Automation Platform to the manifest, and then export the manifest to enable you to download the manifest file that you just created.

Uploading the manifest is shown in Figure 3-18.

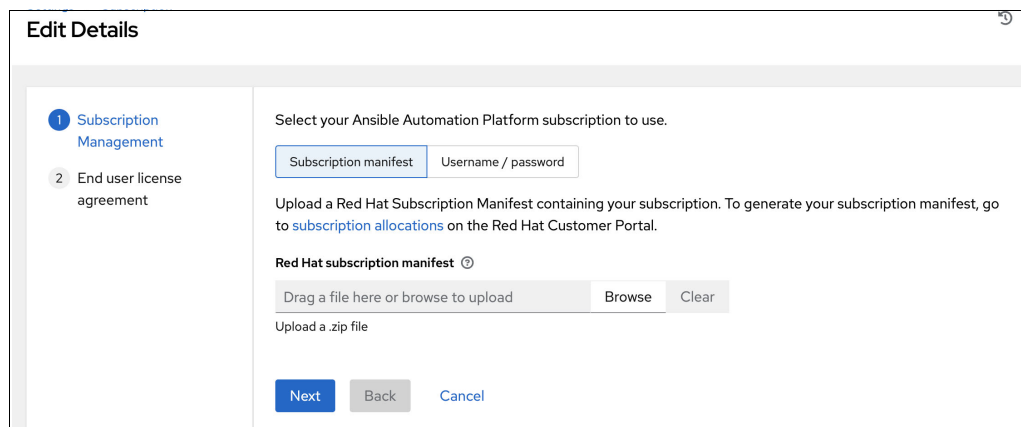


Figure 3-18 Ansible Automation Platform subscription activate - using manifest

More information on creating and using a Red Hat Satellite manifest can be found at <https://www.redhat.com/en/blog/how-create-and-use-red-hat-satellite-manifest>.

If the system is directly connected to the internet, you can use a Red Hat software subscription username and password for the activation as seen in Figure 3-19.

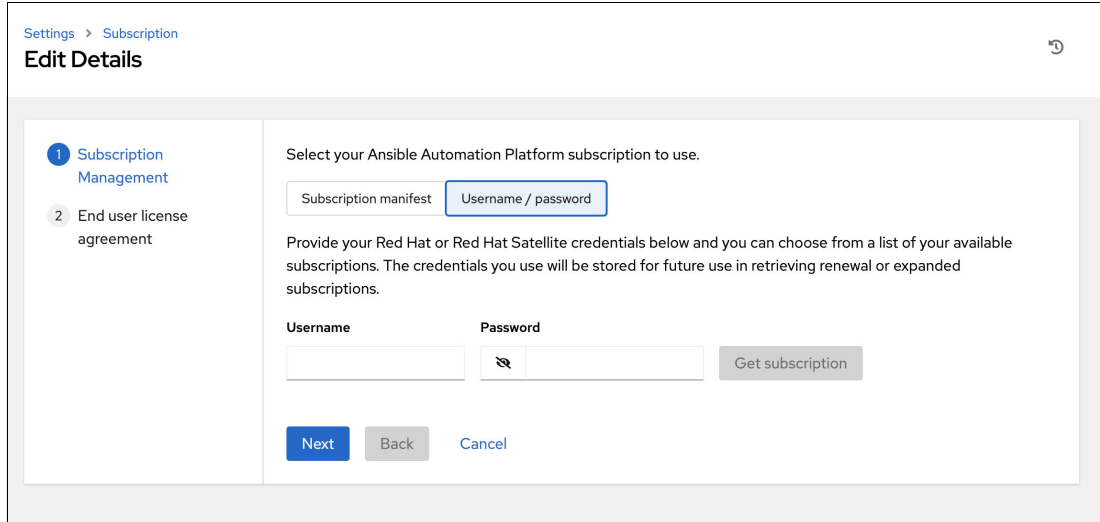


Figure 3-19 Ansible Automation Platform subscription activate - using username password

7. Once you login you need to select the appropriate subscription from the list. An example is shown in Figure 3-20. Click the **Next** button on the User and Automation Analytics screen and finally click the **Submit** button on the End User License Agreements screen.

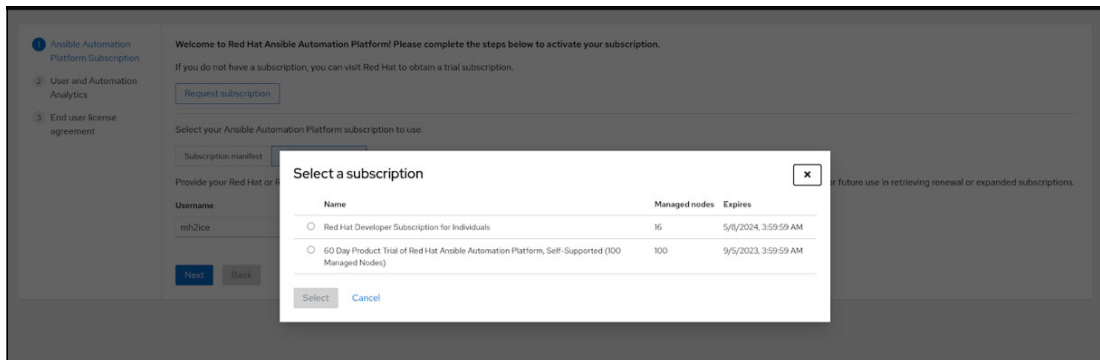


Figure 3-20 Select a subscription for installation

8. Now the Red Hat Ansible Automation Platform is ready for further integration and configuration for you to start automating your environment. See Figure 3-21 on page 127 for details.

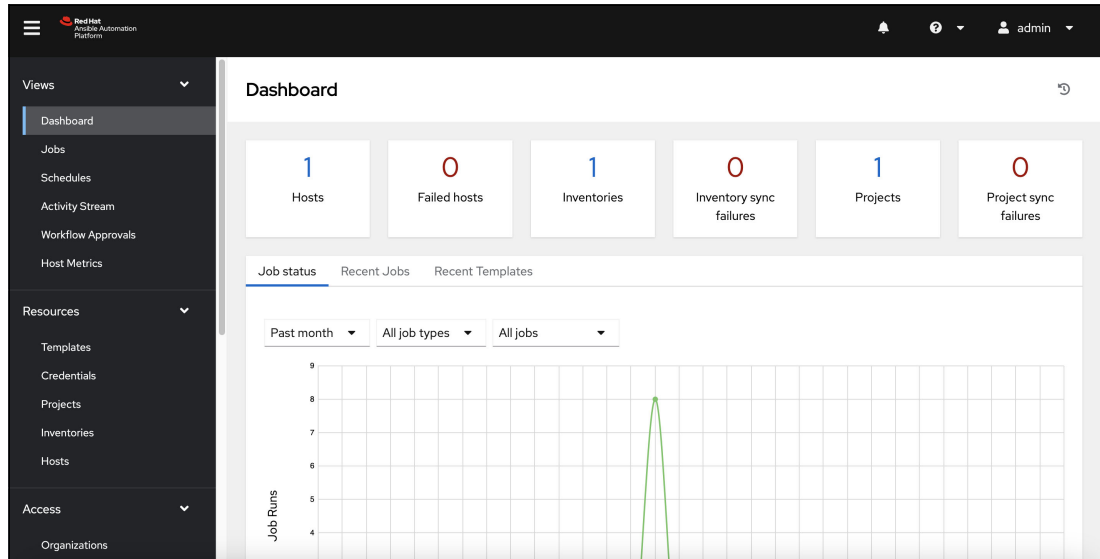


Figure 3-21 Ansible Automation Platform dashboard page.

Further configuration steps

Now that Ansible Automation Platform is installed, it can be used to configure the required integrations and required resources for your automation projects using a web console.

Some of the resources that can be created in the Ansible Automation Platform are:

- Templates (see [Job Templates](#) and [Workflow Job Templates](#))
- Credentials
- Projects
- Inventories
- Hosts
- Organizations
- Users
- Teams

Also you can configure and integrate third party services that you require. Some example services are:

- **Enhanced and Simplified Role-Based Access Control and Auditing:** Configure role-based access control (RBAC). Automation controller allows for the granting of permissions to perform a specific task (such as to view, create, or modify a file) to different teams or explicit users through RBAC.
- **Backup and Restore:** The ability to backup and restore your system has been integrated into the Ansible Automation Platform setup playbook, making it easy for you to backup and replicate your instance as needed. Configure `cron` jobs and use `setup.sh` script for backup and restore.
- **Integrated Notifications:** Configure stackable notifications for job templates, projects, or entire organizations, and configure different notifications for job start, job success, job failure, and job approval (for workflow nodes). Notifications can be integrated with Email, Grafana, Slack or other tools.
- **Authentication Enhancements:** The Automation controller supports LDAP, SAML, token-based authentication. Configure a feasible authentication method.

- **Workflow Enhancements:** In order to better model your complex provisioning, deployment, and orchestration workflows, automation controller expanded workflows in a number of ways:
 - Inventory overrides for Workflows.
 - Convergence nodes for Workflows.
 - Workflow Nesting.
 - Workflow Pause and Approval.
- **Secret Management System:** With a secret management system, external credentials are stored and supplied for use in the automation controller so don't have to provide them directly.
- **Manage playbooks using source control:** Managing playbooks and playbook directories by either placing them manually under the Project or placing the playbooks into a supported source code management (SCM) system, including Git, Subversion, and Mercurial. Configuration and integration with SCM.

For more details on post configuration refer to the [Automation Controller User Guide v4.4](#).

Ansible Automation Controller installation on RHEL

The heart of automation is the Ansible Automation Controller. This is a system which will run the Ansible commands and playbooks in a deterministic way to allow you to automate your environment.

System preparation on RHEL 8

Create a User (e.g. ansible) which will own the environment and install the necessary packages on the OS. Also avoid installing pip packages outside a Python virtualenv, so the OS managed python modules installed for other uses will not be interfered with based on the Python requirements of your Ansible installation.

Also, remember to modify your VIM configuration to replace “**Tab**” with an indent using “2 whitespaces”.

Run the commands shown in Example 3-11 as “root”.

Example 3-11 Install Ansible

```
useradd -m -c "Ansible Controller venv User" ansible
dnf install ansible-core python3-virtualenv vim
```

```
cat << 'EOF' > /etc/pip.conf
[install]
require-virtualenv = true
```

```
[uninstall]
require-virtualenv = true
EOF
```

Create the virtual environment

The following steps help you create the virtual environment.

1. Create the virtual environment using this command:

```
virtualenv --python='/usr/bin/python3.9' ~/venv
```

- Upgrade pip, and other necessary Python libraries, and install requirements for the most used Ansible collections as shown in Example 3-12.

Example 3-12 Install Python libraries

```
python3.11 -m venv ~/venv
source ~/venv/bin/activate
export PYTHONPATH=$( ls -ld ~/venv/lib/python*/site-packages )
pip install -U pip setuptools psutil
pip install jmespath netaddr pywinrm pypsrp pyopenssl

ansible-galaxy collection install community.general ansible.windows ansible.posix
ansible.utils
```

- Create a default `ansible.cfg` for this environment, and configure the default `hosts.ini`, and populate it with at least the Ansible controller itself (`localhost`). See Example 3-13.

Example 3-13 Create ansible.cfg

```
ansible-config init --disabled -t all > ~/ansible.cfg
perl -pi -e "s|^;(inventory=).*|\1~/hosts.ini|g" ~/ansible.cfg

cat << 'EOF' >> ~/hosts.ini
localhost ansible_connection=local
EOF
```

- To make it convenient to get into the virtualenv while logging in as user `ansible`, the source and export lines can be added to the `.bashrc` (or `.profile`) of the user.
- Adjust `vimrc` to make VIM recognize the `yaml/yml` indentation. This is shown in Example 3-14.

Example 3-14 Adjust VIM for yaml notation

```
cat << 'EOF' >> ~/.vimrc
autocmd FileType yaml setlocal ts=2 sts=2 sw=2 expandtab
autocmd FileType yml setlocal ts=2 sts=2 sw=2 expandtab
EOF
```

- Adjust `.bashrc` to reflect the environment as shown in Example 3-15.

Example 3-15 Adjusting .bashrc for Python

```
~/bashrc
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi

# User specific environment
if ! [[ "$PATH" =~ "$HOME/.local/bin:$HOME/bin:" ]]
then
    PATH="$HOME/.local/bin:$HOME/bin:$PATH"
fi
export PATH

export HISTSIZE=100000 # big big history
export HISTFILESIZE=100000 # big big history

# Uncomment the following line if you don't like systemctl's auto-paging feature:
```

```
# export SYSTEMD_PAGER=

# User specific aliases and functions
alias view="vim -R"

source ~/ansible-venv/bin/activate
export PYTHONPATH=$( ls -ld ~/ansible-venv/lib/python*/site-packages )
```

Your virtual environment is now ready to use.

3.3.2 AIX as an Ansible controller

As we have discussed earlier, there are multiple implementation of Ansible that can be used, depending on your requirements and your environment. Ansible Core is the supported implementation for AIX, as Ansible Automation Platform is not supported on AIX. To get the full benefit of Ansible Automation Platform running on your IBM Power server you would need to choose an LPAR running Red Hat Enterprise Linux as your controller.

If you are have previously installed open source tools on AIX, the installation of Ansible will be familiar and will look similar to the installation in a Linux on Power LPAR. However, if you have limited experience with open source deployments on AIX, you need to understand how to use the open source installation methodology.

Ansible on AIX is delivered as a part of [IBM AIX Toolbox for Open Source Software](#). All software delivered through IBM AIX Toolbox for open source applications is packaged using RPM format which is the same format used in Linux. As this is not the AIX-native BFF package format the installation procedure is slightly different.

It is highly recommended that you use *dnf* to install any open source tools in your AIX environment. The *dnf* command in a package manager for RPM packages. This is an updated version of the *yum* command which you may have seen in a Linux environment. While RPM files can be installed without a package manager, using *dnf* has two big advantages – it can automatically resolve dependencies and then install them from package repositories. Without the *dnf* package manager, you would be forced to manually determine any package dependences and then install them. You can find a useful and detailed guide on to installing *dnf* [here](#).

In our testing scenarios, we ran the installs on both AIX 7.2 and AIX 7.3. The process applies to both versions.

Installing on a system with internet connectivity

If your AIX Server has a direct internet connection, just running the *dnf_aixtoolbox.sh* script should install *dnf* on your machine. Download the script from IBM's site and run it using the command */dnf_install.sh -y*. It will run for a while and setup *dnf* if everything is successful.

Example 3-16 shows the steps to install the package.

Example 3-16 Installing DNF on AIX

```
# /usr/opt/per15/bin/lwp-download
https://public.dhe.ibm.com/aix/freeSoftware/aixtoolbox/ezinstall/ppc/dnf_aixtoolbox.sh
# chmod +x dnf_aixtoolbox.sh
# ./dnf_aixtoolbox.sh -y
```

The script downloads newest rpm.rte package and a bundle of RPM packages to be installed. There are many packages in the bundle but the most important for our case are Python 3.9 and DNF itself.

Installing on a system without internet connectivity

If you do not have an internet connection, then some additional steps are required to install *dnf*. The following tips will help you successfully install the *dnf* package:

1. First verify you have your proxy setup correctly, exporting the variables as shown in Example 3-17.

Example 3-17 Setting up proxy variables

```
export http_proxy=http://user:password@IP:PORT/
export https_proxy=http://user:password@IP:PORT/
Your proxy setup should look something like
export http_proxy=http://atilio:b01s111ud0@192.168.0.45:8080/
```

2. The command to run the script is `/dnf_install.sh -y`.
3. The script requires ftp access to IBM, don't worry if you cannot support ftp. Comment out lines 179 to 254 in the script and download the packages manually from one of the following repositories:
 - https://public.dhe.ibm.com/aix/freeSoftware/aixtoolbox/eziinstall/ppc/dnf_bundle_aix_71_72.tar
 - https://public.dhe.ibm.com/aix/freeSoftware/aixtoolbox/eziinstall/ppc/dnf_bundle_aix_73.tar
4. Check your *openssl* version, the script requires *ssl* to be at least at 1.1, We used *openssl-1.1.2.2200.tar.Z* that we downloaded from the link bellow:
 - https://www.ibm.com/resources/mrs/assets/DirectDownload?source=aixbp&lang=en_US

Note: You will need an IBM ID to download this file.

5. If install takes forever it might be failing at `rpm.rte install`, open the tar and extract `rpm.rte` and update it through `smit` or `installp` from the cli.

Alternate installation steps for systems without internet connectivity

If you don't have Internet access on your IBM AIX box, and don't want to follow the steps above, you can download the latest bundle manually to any server and then transfer it to your AIX box.

You can find the latest bundles [here](#). the steps to follow are:

1. There are two bundles – one for AIX 7.1 and 7.2 and another one for AIX 7.3. Choose the correct bundle for your version of IBM AIX.
2. After downloading the bundle, unpack it to a temporary directory.
3. In the temporary directory you will find the script `dnf_install.sh`. Run `./dnf_install.sh -y`. It will run for a while and setup *dnf* if everything is okay.

Steps to take after installation of dnf

After you have installed *dnf* it is recommended to do the following steps:

1. Update the packages by running `dnf -y update`.
2. In the default configuration *dnf* tries to download package information from the IBM's site. If you work in an air-gapped environment without direct access to the Internet:
 - Create local repositories by mirroring IBM repositories.
 - After creating local mirrors, you must reconfigure *dnf* by manually editing `/opt/freeware/etc/dnf/dnf.conf`.

Ansible Installation on AIX

With *dnf* installed and setup, we can proceed to the installation of Ansible, and the `ansible-core` packages.

If you search for Ansible in the repositories, you find three references to it as seen in Example 3-18.

Example 3-18 Searching for Ansible in AIX repositories

```
# dnf search ansible
===== Name & Summary Matched: ansible =====
ansible.noarch : Curated set of Ansible collections included in addition to ansible-core
===== Name Exactly Matched: ansible =====
ansible.ppc : SSH-based configuration management, deployment, and task execution system
===== Name Matched: ansible =====
ansible-core.noarch : A radically simple IT automation system
```

The package you want to install is called `ansible.noarch`. The package `ansible-core.noarch` is the base package of Ansible, providing Ansible Core 2.14.2 at the time of writing this publication.

The package `ansible.noarch` provides some additional collections you usually need to work with Ansible. If you install the package `ansible.noarch`, it will automatically install the package `ansible-core.noarch`.

Important: Do not install the `ansible.ppc` – it is an old version of Ansible.

Example 3-19 shows the command to install Ansible and the resulting output.

Example 3-19 Installing Ansible on IBM AIX

```
#dnf -y install ansible.noarch
Dependencies resolved.
=====
Package                Architecture      Version           Repository        Size
=====
Installing:
ansible                noarch            7.2.0-1          AIX_Toolbox_noarch 47 M
Installing dependencies:
ansible-core           noarch            2.14.2-1         AIX_Toolbox_noarch 3.5 M
python3.9-packaging    noarch            19.2-2           AIX_Toolbox_noarch 58 k
python3.9-pyparsing    noarch            2.4.4-2          AIX_Toolbox_noarch 196 k
python3.9-resolve-lib  noarch            0.5.4-1          AIX_Toolbox_noarch 30 k
=====
Transaction Summary
=====
Install 5 Packages
```

Post installation configuration suggestions

Consider the following configuration suggestions for your Ansible environment.

1. All RPM packages from AIX Toolbox applications are installed in `/opt/freeware`. Usually this directory is not added to the `PATH` variable in `/etc/environment` or to the user's profile. This means that in order to run the Ansible commands you must specify the full path to the command as shown in this command:

`/opt/freeware/bin/ansible-playbook` or `/opt/freeware/bin/ansible-galaxy`.

For your convenience it is recommended to add `/opt/freeware/bin` into the `PATH` variable in your profile as shown in Example 3-20. After you change your profile, you must re-login to enable the changes.

Example 3-20 Adding /opt/freeware/bin to PATH variable

```
# echo 'export PATH=$PATH:/opt/freeware/bin' >> ~/.profile
```

2. The global configuration of Ansible can be found in `/opt/freeware/etc/ansible`. By default, this global configuration is used, but you can set up Ansible to use local project-specific configuration files.

The same applies to Ansible collections. You can install them globally into */usr/share/ansible/collections*, locally for your user, or just for one project.

Note: We recommend that you create configuration files and install collections on a project basis.

3. Make sure the correct locale files are installed. Ansible requires the UTF - 8 locale. The command to validate your installed locale files is shown in Example 3-21.

Example 3-21 Validating the locale files installed

```
# ls | grep -l | grep -i bos.loc
bos.loc.com.utf          7.2.0.0  COMMITTED  Common Locale Support - UTF-8
bos.loc.utf.EN_US       7.2.0.0  COMMITTED  Base System Locale UTF Code
# This is a sample for aix 7.2 the important thing is package, version might change.
```

Note: Not having the correct locale file will cause Ansible commands to fail with the following error:

```
# /opt/freeware/bin/ansible
ERROR: Ansible requires the locale encoding to be UTF-8; Detected ISO8859-1.
```

4. You need to set your user environment as shown in Example 3-22.

Example 3-22 User environment setup for using Ansible

```
# vi .profile
"profile" 8 lines, 309 characters
export PATH=$PATH:/opt/freeware/bin
export TERM=aixterm
LC_MESSAGES=%l.%c
export LC_MESSAGES
NLSPATH=/usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/%l.%c/%N:/usr/lib/nls/msg/%L/%N.
cat:/usr/lib/nls/msg/%l.%c/%N.cat:/usr/lib/nls/msg/%l.%c/%N:/usr/lib/nls/msg/%l.
%c/%N.cat
export NLSPATH
LANG=EN_US
export LANG
```

Running a validation command

Once you are done with all the steps in “Post installation configuration suggestions” on page 132, you can run an Ansible command to validate that the installation of Ansible has completed successfully. You can run the command shown in Example 3-23 to validate the installation and display the version of Ansible that is installed.

Example 3-23

```
# ansible --version
ansible [core 2.14.2]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/usr/share/ansible/plugins/modules',
'/usr/share/ansible/plugins/modules']
  ansible python module location =
/opt/freeware/lib/python3.9/site-packages/ansible
  ansible collection location =
./usr/share/ansible/collections:/usr/share/ansible/collections
  executable location = /opt/freeware/bin/ansible
```

```
python version = 3.9.16 (main, Jun 28 2023, 12:45:03) [GCC 8.3.0]
(/opt/freeware/bin/python3.9)
  jinja version = 3.0.3
  libyaml = True
```

As you can see from the example, Ansible core Version 2.14.2 was installed and the Python version being used is Python3 Version 3.9.16.

Files installed during Ansible installation on AIX

Once the Ansible Core rpm installation completes in the system, the system will have the configuration file and binaries. The commonly used files are shown in Table 3-3.

Table 3-3 List of files that associated with ansible-core rpm

Important Files and Executables	Description
/etc/ansible/ansible.cfg	Default configuration file that comes with RPM packages and it will have all the required settings, the location of module search path, module, executable, inventory file and so on. But the configuration file will follow the presidency based on its location. (Linked to <i>/opt/freeware/etc/ansible/ansible.cfg</i>)
/etc/ansible/hosts	Sample inventory for the managed node or target host where automation tasks will be executed. (Linked to <i>/opt/freeware/etc/ansible/hosts</i>)
/opt/freeware/bin/ansible-config	<p>Command to view the effective Ansible configuration details and the file location. Because configuration file has the presidency base on location, as below:</p> <ul style="list-style-type: none"> ▶ <i>/etc/ansible/ansible.cfg</i> - Default configuration file, used if present ▶ <i>~/.ansible.cfg</i> - User configuration file, overrides the default config if present ▶ <i>./ansible.cfg</i> - Local configuration file that is located in the current working directory assumed to be 'project specific' and overrides the rest if present. ▶ <i>ANSIBLE_CONFIG</i> - Specify override location for the ansible config file <p>For example the command below can create a sample configuration for you.</p> <pre># ansible-config init --disabled -t all > ansible.cfg</pre>
/opt/freeware/bin/ansible	<p>Command to define and run a single task 'playbook' against a set of hosts.</p> <p>For example: Run the shell module to execute a command</p> <pre># ansible all -i hosts -m shell -a "hostname"</pre>
/opt/freeware/bin/ansible-console	<p>Command that allows for on-the-fly execution of Ansible modules or arbitrary commands to the hosts.</p> <p>For example:</p> <pre># ansible-console -i hosts --limit all -u root</pre>

Important Files and Executables	Description
/opt/freeware/bin/ansible-doc	Command to display information on specific modules installed in Ansible libraries. For example: To check copy module documentation. # ansible-doc copy
/opt/freeware/bin/ansible-playbook	Command to run Ansible playbooks, executing the defined tasks on the targeted hosts. For example: To run a sample playbook. # ansible-playbook myplaybook.yml
/opt/freeware/bin/ansible-vault	Command that can be used as an encryption/decryption utility for Ansible that can encrypt any structured data file used by Ansible.

Next Steps

Now that you have installed Ansible, the next steps are to set up the configuration appropriate for your environment. The following steps are recommended:

1. Generate the configuration file using the command below:

```
# ansible-config init --disabled -t all > ansible.cfg
```

2. Display the effective configuration and the configuration file location, in terms of current working location or directory as shown in Example 3-24.

Example 3-24 Display effective configuration

```
# ansible --version
ansible [core 2.14.2]
  config file = /ansible.cfg
  configured module search path = ['./.ansible/plugins/modules',
'/usr/share/ansible/plugins/modules']
  ansible python module location = /opt/freeware/lib/python3.9/site-packages/ansible
  ansible collection location = /.ansible/collections:/usr/share/ansible/collections
  executable location = /opt/freeware/bin/ansible
  python version = 3.9.16 (main, Jun 28 2023, 12:45:03) [GCC 8.3.0]
(/opt/freeware/bin/python3.9)
  jinja version = 3.0.3
  libyaml = True
```

3. Verify the inventory file name and location as shown in Example 3-25.

Example 3-25 Verify inventory file location

```
# grep -vE "^#|^;" /etc/ansible/ansible.cfg|grep -v ^$
[defaults]
[privilege_escalation]
[persistent_connection]
[connection]
[colors]
[selinux]
[diff]
[galaxy]
[inventory]
[netconf_connection]
[paramiko_connection]
[jinja2]
```

```
[tags]
[runas_become_plugin]
[su_become_plugin]
[sudo_become_plugin]
[callback_tree]
[ssh_connection]
[winrm]
[inventory_plugins]
[inventory_plugin_script]
[inventory_plugin_yaml]
[url_lookup]
[powershell]
[vars_host_group_vars]
```

4. Verify the inventory file configuration as shown in Example 3-26.

Example 3-26 Verify inventory configuration

```
atilio-ansiblerh73: /> cat /etc/ansible/hosts
# This is the default ansible 'hosts' file.
#
# It should live in /etc/ansible/hosts
#
# - Comments begin with the '#' character
# - Blank lines are ignored
# - Groups of hosts are delimited by [header] elements
# - You can enter hostnames or ip addresses
# - A hostname/ip can be a member of multiple groups
[linux]
ansible-AAP-redbook
hugo-rhel8-ansible
vasco-rhel8-ansible
revelez-rhel9-ansible
[aix]
vitorio-ansibleaix72
atilio-ansibleaix73
```

5. Run an ad-hoc command to verify the functionality as shown in Example 3-27.

Example 3-27 Test Ansible functionality with ad-hoc command (sshpass must be installed with dnf)

```
atilio-ansiblerh73: /> ansible all -i hosts -m shell -a "hostname" -u root -k
SSH password:
hugo-rhel8-ansible | CHANGED | rc=0 >>
hugo-rhel8-ansible
```

Additional Preparation and Configuration of Ansible on AIX

It is suggested that a separate user be created to manage automation activities, generate ssh keys for managed nodes access, create a virtual environment for specific python versions, set environment variables for better working environment and otherwise manage your environment. Follow these steps to setup the user (we used the user name *ansible*):

- ▶ Create a user called ansible using the following command:

```
# vitorio-ansibleaix72: /> mkuser -a "gecos=Ansible Controller User" ansible
```

- ▶ Install any additional python libraries or modules depending on requirements. Use the following command:

```
# dnf install python3-pip
```

Note: If any specific python libraries or modules are not available or not shipped with AIX OS in rpm format, they can be installed via **pip** (the python packages manager). You can create a virtual environment, like a virtual machine or Linux chroot, that will have an isolated structure of lightweight directories separated from the actual operating system python directories to allow you to use different versions of python modules, files, or configurations.

- ▶ Create a `~/vimrc` file to customize the vim editor configuration to use the 2 space indentation for yaml file editing as shown in Example 3-28.

Example 3-28 Modify vim editor configuration

```
# cat << 'EOF' >> ~/.vimrc
autocmd FileType yaml setlocal ts=2 sts=2 sw=2 expandtab
autocmd FileType yml setlocal ts=2 sts=2 sw=2 expandtab
EOF
```

- ▶ Generate and copy the ssh key from the Ansible Automation Controller node to the managed nodes using the following commands:

```
# ssh-keygen
# ssh-copy-id root@192.168.xxx.xxx
```

Now your AIX based Ansible controller is ready for use for automation tasks.

Installing the IBM Power AIX Collection

We discussed the IBM Power Systems AIX collection in “IBM Power Systems AIX collection” on page 39. The AIX collection contains a large number of modules and roles to help you manage your IBM AIX LPARs. The collection can be installed on your Ansible controller node using the **ansible-galaxy** command as shown in Example 3-29.

Example 3-29 Installing ibm.power_aix collection

```
$ ansible-galaxy collection install ibm.power_aix
Starting galaxy collection install process
Process install dependency map
Starting collection install process
Downloading https://galaxy.ansible.com/download/ibm-power_aix-1.6.4.tar.gz to
/home/ansible/.ansible/tmp/ansible-local-13042144f894hz11/tmpvmhd3sw5/ibm-power_ai
x-1.6.4-x64201pn
Installing 'ibm.power_aix:1.6.4' to
'/home/ansible/.ansible/collections/ansible_collections/ibm/power_aix'
ibm.power_aix:1.6.4 was installed successfully
```

If you don't have direct access, you can download the collection on another server and then copy it to your Ansible controller node as shown in Example 3-30.

Example 3-30 Installing ibm.power_aix collection from local file

```
$ ls
ibm-power_aix-1.6.4.tar.gz
$ ansible-galaxy collection install ibm-power_aix-1.6.4.tar.gz
Starting galaxy collection install process
```

```

Process install dependency map
Starting collection install process
Installing 'ibm.power_aix:1.6.4' to
'/home/ansible/.ansible/collections/ansible_collections/ibm/power_aix'
ibm.power_aix:1.6.4 was installed successfully

```

The collection documentation has a demo inventory file that you can look at. However, for our test environment our inventory file looks like what is shown in Example 3-31.

Example 3-31 Our inventory file for our test environment

```

all: # keys must be unique, i.e. only one 'hosts' per group
  hosts:
  vars:
  children: # key order does not matter, indentation does
    aix:
      children:
        nimserver:
          hosts:
            narancio-nim-master:
              ansible_host: narancio-nim-master
          vars:
            vm_targets: vitorio-ansibleaix72
        nimclient:
          hosts:
            vitorio-ansibleaix72:
              ansible_host: vitorio-ansibleaix72
            atilio-ansibleaix73:
              ansible_host: atilio-ansibleaix73
          vars:
            res_group: basic_res_grp
      hosts:
        cascarilla-ansibleaix73:
          ansible_host: cascarilla-ansibleaix73
    vios:
      hosts:
        gpc-s924-vios1:
          ansible_host: gpc-s924-vios1
      vars:
        res_group: vios_res_grp

```

Using a NIM server as your Ansible controller

The Ansible Power AIX collection has a large number of modules. As you look through the list of modules as seen in Table 1-7 on page 40, you will notice that there are several that prerequisite the use of the NIM server. This reinforces the recommendation made earlier to consider running your Ansible controller node for your AIX LPARs on your NIM server. When you run the NIM based modules on your NIM server, then the Ansible controller will be able to run commands on the NIM clients, and will already have the required contents to support upgrades or installations for your AIX Ansible Clients.

3.3.3 IBM i as an Ansible controller

Ansible, a powerful automation tool, is enhanced on the IBM i platform through the integration of key components, each serving a crucial role in enabling automation processes.

PASE (Portable Application Solutions Environment)

PASE, integrated within IBM i, offers a runtime environment that facilitates the execution of chosen applications. This environment includes industry-standard and defacto standard shells, establishing a robust scripting platform. Functioning as an AIX release, PASE can be customized for communication with System Licensed Internal Code (SLIC), utilizing memory from SLIC, which is also utilized by the Integrated Language Environment (ILE). PASE and ILE work together on IBM i. However, Ansible necessitates Python installation and operation from PASE, while playbooks require integration of Python commands.

Red Hat Package Manager (RPM)

RPM hosts precompiled binary files, with IBM crafting versions for the IBM i platform. Stored within IFS on IBM i, RPM files adhere to the format `<name>-<version>-<release>.<os>.<architecture>.rpm`. A sample RPM filename is `Ansible-2.9.9-1.ibm72.noarch.rpm`.

RPM files for IBM i are accessible at [IBM i software](#).

Yellowdog Updater, Modified (YUM)

YUM, a free open-source utility, aids package management through command-line interactions. YUM allows the installation or update of RPM packaged software and handles dependencies within the RPM package. IBM i utilizes YUM to install, update, upgrade, and remove packages.

It is important to install YUM before initiating Ansible. This installation can be performed with or without an Internet connection. To install YUM without an Internet connection, a one-time bootstrap process is used. More information can be found at [YUM installation](#).

After YUM is installed, navigate to the directory where it resides, issue `cd /Qopensys/pkgs/bin/`, then `yum -h`.

Note: Packages can be managed through SSH terminal commands or IBM Access Client Solution.

SSHPass and ssh-keygen

The SSHPass package, non-interactive, acts as an Ansible Controller on IBM i. It simulates an interactive user entering the required SSH connection password. Use the following command to install SSHPass: `yum install sshpass`

Additionally, ssh-keygen is a vital component of SSH. It generates a secure connection between the Ansible controller and IBM i endpoints' systems, employing a pair of keys - public and private. To generate these keys, execute the command: `ssh-keygen -t rsa`

Note: By establishing these components and their integration, Ansible on IBM i gains a powerful foundation, ready for versatile automation and management tasks.

Ansible Controller installation on IBM i

Installing the Ansible Controller on the IBM i platform involves setting up the RPM environment. Ensure that RPM is configured by following the steps outlined in the [IBM i Open Source documentation](#).

There are three methods for installing Ansible Controller:

1. **Installation without internet access:** If your IBM i system lacks direct Internet access, you can manually download the RPM package from the [IBM repository](#). To facilitate this, employ a proxy server on a local intranet. Configure YUM to utilize the proxy by editing the `yum.conf` file located at `/Q0penSys/etc/yum/yum.conf`. Add the proxy settings as shown in Example 3-32.

Example 3-32 Configuring proxy for YUM

```
[main]
proxy=http://proxy.mycompany.example.com:1234
proxy_username=user_name
proxy_password=passw0rd
```

If desired, you can create a local repository using `reposync` and `createrepo`, generating a complete copy of the remote repository. Details on this process can be found [here](#).

2. **Internet Access from IBM i system:** If your IBM i system has Internet access, ensure that Python v3.6+ is installed. Log in to a SSH terminal and execute `yum install ansible`. Next execute `ansible -version`. The first step installs Ansible and the second allows you to verify the installation by checking its version.
3. **Installation using IBM i Access Client Solutions:** Ansible can be conveniently installed through IBM ACS. For detailed step-by-step instructions, please refer to the section titled [Installation using IBM i Access Client Solutions \(ACS\)](#). This approach offers a user-friendly method to set up Ansible on your IBM i platform.

Configuring Ansible for IBM i

The configuration of Ansible is a crucial step to ensure its effective functionality. Here is a step-by-step guide on how to configure Ansible on your IBM i platform:

1. **Locate the configuration file:** The Ansible configuration file, typically named `ansible.cfg`, is usually present in the `/etc/ansible` directory. However, you can also create this file in a different path or directory if needed.
2. **Parameter flexibility:** The `ansible.cfg` file encompasses a wide range of parameters that can be utilized. It is important to note that activating all parameters is not mandatory, and you can configure them as required for your specific setup.
3. **Creating the configuration file:** To create the Ansible configuration file, issue:

```
vi /etc/ansible/ansible.cfg.
```

In this file, you can specify various configurations to tailor Ansible's behavior to your needs. One of the commonly customized sections is `[defaults]`.

Example 3-33 shows how to configure the `ansible.cfg` file for IBM i:

Example 3-33 Parameters for the Ansible configuration file for IBM i

```
[defaults]
inventory =
~/ansible/collections/ansible_collections/ibm/power_ibmi/playbooks/hosts_ibmi.ini
library = ~/ansible/collections/ansible_collections/ibm/power_ibmi/plugins/action
```

Note: Part of the Ansible configuration process involves setting up the `ansible.cfg` file. While the default location for this file is typically `/etc/ansible` on various platforms, including Ansible Controller on IBM i, it is important to create one if it does not exist.

For a comprehensive list of parameters that can be included in an Ansible configuration file, you can refer to [this document](#). This reference provide you with a deeper understanding of the various options available for configuring Ansible to suit your requirements.

Post-Installation configuration steps on IBM i

After the installation of Ansible, further configuration is necessary to effectively utilize Ansible collections, modules, and automation capabilities on IBM i. Ansible collections represent the modern standard for distributing and maintaining automation components, encompassing modules, action plugins, roles, and sample playbooks.

Note: Installing collections with Ansible-Galaxy is supported only in Ansible 2.9+

The following procedure guides you through the process of configuring Ansible Galaxy, a repository for Ansible on IBM i which contains content from the broader Ansible community.

1. Install the IBM i collection from Ansible Galaxy, the designated package manager for Ansible, Example 3-34 displays the command.

Example 3-34 Command to install Ansible Galaxy collections

```
# ansible-galaxy collection install ibm.power_ibmi
Process install dependency map
Starting collection install process
Installing 'ibm.power_ibmi:1.5.0' to
'/HOME/QSECOFR/.ansible/collections/ansible_collections/ibm/power_ibmi'
```

2. Check the installation path of the collections in the IFS (Integrated File System) using the `cd` command, issue:

```
cd /home/qsecofr/.ansible/collections/ansible_collections/ibm/power_ibmi
```

3. Display the content of `power_ibmi` directory using the long listing command `ls -l`

Example 3-35 Displays the content of power_ibmi directory

```
# ls -l
total 220
-rw-r--r-- 1 qsecofr 0 90760 Jul 20 23:33 FILES.json
-rw-r--r-- 1 qsecofr 0 1241 Jul 20 23:33 MANIFEST.json
-rw-r--r-- 1 qsecofr 0 1284 Jul 20 23:33 README.md
-rw-r--r-- 1 qsecofr 0 186 Jul 20 23:33 bindep.txt
drwxr-sr-x 3 qsecofr 0 8192 Jul 20 23:33 changelogs
drwxr-sr-x 4 qsecofr 0 8192 Jul 20 23:33 docs
drwxr-sr-x 2 qsecofr 0 8192 Jul 20 23:33 meta
drwxr-sr-x 5 qsecofr 0 28672 Jul 21 12:20 playbooks
drwxr-sr-x 5 qsecofr 0 8192 Jul 20 23:33 plugins
drwxr-sr-x 24 qsecofr 0 28672 Jul 20 23:33 roles
drwxr-sr-x 9 qsecofr 0 12288 Jul 20 23:33 usecases
```

4. Navigate back to the user's home directory by issuing the following command:

```
cd /home/qsecofr
```

5. Create a `.ssh` directory in the user's home directory, issue the command:

```
mkdir -p /home/qsecofr/.ssh
```

6. Verify the creation of the new directory as seen in Example 3-36.

Example 3-36 Displays the ssh directory

```
# ls -la
total 56
drwxr-sr-x 5 qsecofr 0 8192 Sep 29 23:15 .
drwxrwsrwx 5 qsys 0 8192 Sep 20 2020 ..
drwx--S--- 3 qsecofr 0 8192 Sep 20 22:35 .ansible
drwxrwsrwx 3 qsecofr 0 8192 Oct 20 2019 .java
-rw-r--r-- 1 qsecofr 0 42 Oct 20 2019 .profile
drwxr-sr-x 2 qsecofr 0 8192 Sep 17 23:15 .ssh
-rw----- 1 qsecofr 0 16 Oct 20 2019 .vi_history
```

7. Generate an SSH key pair for the Ansible controller on IBM i and its managed hosts. Enter the following command, pressing **Enter** three times without providing a passphrase or changing the default location of the key. The results are shown in Example 3-37.

Example 3-37 Generating the rsa key pair

```
# ssh-keygen -t rsa -C "10.10.10.10"
Generating public/private rsa key pair.
Enter file in which to save the key (/HOME/QSECOFR/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /HOME/QSECOFR/.ssh/id_rsa.
Your public key has been saved in /HOME/QSECOFR/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:V5VzRDMfVjLMFa7Wh8+V8q6e5IdQbFPys5HYc/tegIA 10.10.10.10
The key's randomart image is:
+---[RSA 3072]-----+
|
|   o+@B|
|  . oB+*|
| E . o *+o|
|   o Bo0o|
|  S . +++oX|
|  . .. o*o|
|   ...=|
|   o.o.o|
|  . =00.|
+-----[SHA256]-----+
```

8. Confirm the generated content using the `ls -la` command as shown in Example 3-38.

Example 3-38 Displaying the public and private rsa key pair

```
# ls -la
total 52
drwxr-sr-x 2 qsecofr 0 12288 Sep 22 20:47 .
drwxr-sr-x 7 qsecofr 0 24576 Sep 22 17:31 ..
-rw----- 1 qsecofr 0 2602 Sep 22 20:47 id_rsa
-rw-r--r-- 1 qsecofr 0 565 Sep 22 20:47 id_rsa.pub
```

9. Before copying the SSH key to the managed hosts, install **sshpas**, a tool that facilitates password authentication in both interactive and non-interactive modes. Use the following command shown in Example 3-39 to install **sshpas**.

Example 3-39 Installing sshpass via yum

```
# yum install sshpass
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package sshpass.ppc64 0:1.06-1 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package                Arch          Version
Repository              Size
=====
Installing:
 sshpass                ppc64         1.06-1         ibm
30 k

Transaction Summary
=====
Install      1 Package

Total download size: 30 k
Installed size: 77 k
Is this ok [y/N]: Y
Downloading Packages:
sshpass-1.06-1.ibm7.2.ppc64.rpm
| 30 kB 00:00:00
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : sshpass-1.06-1.ppc64
1/1

Installed:
  sshpass.ppc64 0:1.06-1

Complete!
```

IBM i hardware requirements for Ansible Controller

The hardware resources required for Ansible Controller on IBM i depend on factors such as the number of agents served, frequency of operator check-ins, managed resources per agent, and the complexity of manifests and modules in use.

For an IBM i system with Ansible Controller installed, Table 3-4 outlines recommended hardware specifications based on the number of managed nodes:

Table 3-4 Hardware requirements for Ansible Controller on IBM i

Managed Nodes	CPU (cores)	Memory (GB)	Disk (GB)	IBM i Release
Dozens	0.25	2	80	7.3, 7.4, 7.5
1000+	0.5	8	80	7.3, 7.4, 7.5

Note: These hardware requirements ensure optimal performance and efficient management of Ansible tasks and operations on your IBM i system.

3.4 Preparing your systems to be Ansible clients

In general, there is minimal setup required to run the Ansible client on a device or server since Ansible does not require an agent to be installed on the client, and it uses SSH to connect the management node to the client node.

However, there are some basic setup considerations for each of the LPARs that will be Ansible clients. The obvious one is to ensure that SSH is installed and available. All supported operating systems that run on IBM Power support SSH, but there are some considerations for ensuring that it is installed correctly which can vary by operating system. Additionally, Ansible requires Python be installed, and again this process differs depending on the operating system used in your client.

The next sections describe the recommended actions to prepare your Ansible client based on the operating system used.

3.4.1 Linux as Ansible-managed client

Even though we could say that every Linux node, can work as Ansible Client natively, it is necessary to have python3 installed, python3-pip installed, and setup some parameters on the ssh configuration.

The following tasks will help avoid delays caused by dns resolution, or ssh timeouts:

1. In `/etc/ssh/sshd_config`, make the following changes:


```
Uncomment GSSAPIAuthentication no
Uncomment GSSAPICleanupCredentials yes
Uncomment UseDNS No
```
2. Add your Ansible-Core node to the `/etc/hosts` file if no dns is setup
3. Verify your python3 install as shown in Example 3-40.

Example 3-40 Verifying python install

```
[root@hugo-rhel8-ansible ~]# dnf list python3*
Updating Subscription Management repositories.
Installed Packages
python3-asn1crypto.noarch    0.24.0-3.e18           @anaconda
python3-cffi.ppc64le         1.11.5-5.e18           @anaconda
python3-configobj.noarch     5.0.6-11.e18           @anaconda
python3-cryptography.ppc64le 2.3-3.e18              @anaconda
```

4. Create a user for connection or Setup ssh keys if you want to uses passwordless access

3.4.2 AIX as Ansible-managed client

This section describes tips and hints to use AIX as an Ansible-managed client, using just straight ssh or the AIX Collection from galaxy.

First, to avoid delays caused by dns resolution, or ssh timeouts check the points bellow

1. Modify `/etc/ssh/sshd_config`

```
Uncomment GSSAPIAuthentication no
UncommentGSSAPICleanupCredentials yes
Uncomment UseDNS No
```

2. Add your Ansible-Core node to the `/etc/hosts` file if no dns is setup, check your `/etc/netsvc.conf`
3. Create a user for connection (we used ansible) or setup ssh keys if you want to use passwordless access,

Important: From a security prospective it is undesirable to use root for Ansible. We recommend to create a separate user for Ansible. If at some point you must escalate your privileges, you can use `su` with password, `sudo` or AIX-native RBAC to achieve it. `Sudo` is available as a package from the AIX Toolbox for Open Source Software or directly from the site of the author of `sudo` Todd Miller.

4. Verify your python3 install. Considerations for the installation of python3 are discussed in “Python installation considerations”.

Python installation considerations

Python is a primary prerequisite for your AIX Ansible client nodes. The method of installing python will depend on the version of AIX you are using.

To start, you can verify if python3 is installed using the `dnf` command shown in Example 3-41.

Example 3-41 Verify python3 is installed

```
atilio-ansibleaix73:~>dnf list python3*
Last metadata expiration check: 1 day, 3:34:59 ago on September 10, 2023 at 10:17:18 AM -03.
Installed Packages
python3.ppc                    3.9.16-0      @System
python3-dnf.noarch            4.2.17-64_6   @System
python3-gpg.ppc              1.13.1-64_3   @System
python3-hawkey.ppc           0.39.1-64_5   @System
python3-libcomps.ppc         0.1.15-64_1   @System
python3-libdnf.ppc           0.39.1-64_5   @System
python3-librepo.ppc          1.11.0-64_2   @System
python3-pip.noarch            22.2.2-1      @AIX_Toolbox_noarch
python3.9.ppc                 3.9.16-0      @System
```

Starting with AIX 7.3, Python is a standard part of the AIX distribution. You can check if your specific AIX installation has python installed by using `lspp` command shown in Example 3-42 on page 146.

Example 3-42 Checking python installation on AIX 7.3

```
# lsipp -L python3.9.base
Fileset                Level State Type Description (Uninstaller)
-----
python3.9.base         3.9.12.0 C   F   Python 3.9 64-bit binary
                        distribution
```

On AIX versions earlier than 7.3, we recommend the use of python from the AIX Toolbox for Open Source Software. We described the process of DNF installation in the section 3.3.2, “AIX as an Ansible controller” on page 130. Python is installed together with DNF.

Depending on how you installed Python, the main python binary can be either `/opt/freeware/bin/python3` or `/usr/bin/python3`. For the sake of simplicity and standardization, you should choose one standard path to access python3 across your whole environment. If you use `/usr/bin/python3` you will not need to make any other changes in your Ansible playbooks, but you if you choose another location, then your future playbooks or inventories must define the variable `ansible_python_interpreter` with the full path to python3 for your clients.

Wherever you install python, be sure that you have updated your `PATH` settings. It is also recommended that you create links to python in `/usr/bin/` as show in Example 3-43.

Example 3-43 Create link to python in /usr/bin/

```
atilio-ansibleaix73: />ln -s /usr/bin/python3 /usr/bin/python
ln: /usr/bin/python exists. Specify -f to remove.
atilio-ansibleaix73: />ls -lrt /usr/bin/python
lrwxrwxrwx  1 root  system          16 Sep 11 09:05AM /usr/bin/python ->
/usr/bin/python3
atilio-ansibleaix73: />ls -lrt /usr/bin/python3
lrwxrwxrwx  1 root  system          30 Sep 01 09:06AM /usr/bin/python3 ->
/usr/opt/python3/bin/python3.9
```

3.4.3 IBM i as Ansible-managed client

In this section we explore the convergence of IBM i with the Ansible ecosystem as a managed client. This integration empowers efficient system management and configuration using Ansible's automation capabilities. We also explore how IBM i becomes part of the Ansible-managed environment, enhancing operational efficiency and configuration control.

Enabling managed nodes on IBM i

The systems that can be managed by the Ansible Controller are known as IBM i endpoints or managed nodes. To prepare each endpoint system for management, the following requirements need to be met:

1. License Program Products Required:
 - a. IBM Portable Utilities for i (5733-SC1 option base)
 - b. OpenSSH, OpenSSL, zlib functions (5733-SC1 option 1)
 - c. IBM HTTP Server for i (5770-DG1 option base)

Note: To determine if these Licensed Program Products (LPPs) are already installed, you can use the following SQL queries from a 5250 terminal:

STRSQL

```
select * from QSYS2.SOFTWARE_PRODUCT_INFO where product_id = '5733SC1';
select * from QSYS2.SOFTWARE_PRODUCT_INFO where product_id = '5770DG1';
```

If they are not installed, you can download them from [this site](#). For download and installation instructions see this [IBM Support Site](#).

2. Check Open Source packages and ensure that Python 3.6+ is available, issue the following commands:

```
yum search python | grep 3
python --version
```

Note: If they are not installed, install the required Python packages:

```
yum install python3 python3-itoolkkit python3-ibm_db
```

3. To automatically start SSH after an Initial Program Load (IPL), issue the following command:


```
system "chgtcpsvr svrspcval(*sshd) autostart(*yes)"
```
4. Make sure the home directory exists for the user defined as the Ansible user. To create a home directory on the IBM i LPAR to store the user's SSH-related objects, issue the command:


```
mkdir -p /home/<user>/.ssh,
```

 where **<user>** needs to be changed to your user id.
5. Transfer the previously generated public key shown in Example 3-38 on page 142, named **id_rsa.pub**, from the Ansible Controller on IBM i to the managed nodes or endpoints. Example 3-44 illustrates the process.

Example 3-44 Transferring the public key from control node to the managed node

```
# ssh-copy-id qsecofr@192.168.1.100
/QOpenSys/pkgs/bin/ssh-copy-id: INFO: Source of key(s) to be installed:
"/HOME/QSECOFR/.ssh/id_rsa.pub"
The authenticity of host '192.168.1.100 (192.168.1.100)' can't be established.
ECDSA key fingerprint is SHA256:lno5PMGRhgupc23tpStiFRE4cPxVEmpZ/dmN1kfJ1RQ.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/QOpenSys/pkgs/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter
out any that are already installed
/QOpenSys/pkgs/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are
prompted now it is to install the new keys
qsecofr@192.168.1.100's password: "type the password"
sh: test: argument expected
Number of key(s) added: 1
```

Now try logging into the machine, with: "ssh 'qsecofr@192.168.1.100'" and check to make sure that only the key(s) you wanted were added.

6. Ensure that the authorized SSH key has been successfully transferred to the managed node. To confirm, perform a passwordless login to the managed host. Shown in Example 3-45

Example 3-45 Passwordless login to managed host

```
qsecofr@CONTROLLER:~# ssh qsecofr@192.168.1.100

GNU bash, version 4.4.23(1)-release (powerpc-ibm-os400)
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

This shell has been enhanced by .dotfiles version 1.3.0

qsecofr@MANAGED:~#
```

Using the IBM i Ansible collection

The previous section provided information on how to enable an IBM i system as a managed node or client in your Ansible management environment. This section will discuss some additional steps to take and provides some suggestions to make the management of your IBM i clients more effective.

Building an Ansible Inventory

Creating an accurate inventory is fundamental for Ansible's effective management of different managed nodes or endpoints. Follow these guidelines to build a well-structured Ansible inventory:

- ▶ **Managed nodes:** Ansible Controller interacts with various managed nodes or endpoints, each serving a specific purpose in your infrastructure.
- ▶ **Default inventory location:** The default inventory file is named **hosts**, and its location is typically **/etc/ansible/hosts**.
- ▶ **Custom inventory path:** You have the flexibility to specify an alternative inventory file path by using the **-i <path>** option while executing commands.
- ▶ **Multiple inventory files:** Ansible supports using multiple inventory files simultaneously, offering adaptability for complex environments.
- ▶ **Creating an inventory file:** To create your own inventory file, issue:

```
vi /etc/ansible/hosts
```

Example 3-46 shows the content of an inventory file

Example 3-46 Inventory file for IBM i demonstration

```
[local]
localhost ansible_connection=local
[ibmi]
192.168.1.100 ansible_ssh_user=avalosm ansible_ssh_pass=abc123
[ibmi:vars]
ansible_python_interpreter="/Q0pensys/pkgs/bin/python3"
```

- ▶ **IBM i Collections inventory:** Ansible Galaxy's IBM i collections offer a preexisting inventory file located at:

```
~/ansible/collections/ansible_collections/ibm/power_ibmi/playbooks/
```

Note: This inventory can be edited to fit your needs. The file named `hosts_ibmi_ini` in this directory can be utilized for this purpose.

- **Check Inventory configuration:** To verify your current inventory configuration, use the following command:

```
ansible-inventory --list -y
```

Tip: Building a comprehensive inventory ensures that Ansible can effectively manage the specified nodes and endpoints. This foundation is essential for orchestrating automation and configuration tasks across your infrastructure.

Good practice: For the purpose of demonstration and general understanding in this chapter, we have utilized the QSECOFR user profile. However, it is strongly advised against using the QSECOFR user profile in conjunction with SSH on IBM i. This recommendation applies to both the Ansible controller and the managed nodes or endpoints. It is recommended to create a new user profile with comparable authority levels.

For this new user profile, ensure the creation of a HOME directory on the IBM i system. Additionally, configure appropriate permissions for the user's profile HOME directory by executing the `chmod 755` command. Modify the ownership of the HOME directory to match the SSH user. Integrate the HOME directory into the user profile.

In alignment with Ansible's prerequisites for generating and copying public keys, it is crucial to establish a dedicated directory within the user's HOME directory. This directory named `ssh`. Configure permissions for this SSH directory using the `chmod 700` command to ensure the appropriate level of security.

Ansible ad-hoc command

Ansible ad-hoc commands provide an approach to executing essential operations on IBM i systems. Operating from the Ansible controller, these commands enable specific tasks to be carried out efficiently. Each command focuses on a single operation, allowing for quick task execution. If multiple tasks are required, they can be performed through a sequence of commands. For added flexibility, alternative inventory file locations can be specified using the `-i [inventory]` option.

Furthermore, even the Ansible controller or control node itself can be incorporated as a managed host within the inventory file. This functionality enhances the efficiency of executing routine tasks. The subsequent examples shed light on the practical applications of Ansible Ad-Hoc commands:

- Verifying the readiness of all inventory hosts for management by the Ansible controller can be done by running the command shown in Example 3-47.

Example 3-47 Verifying host availability with Ansible Ad-Hoc Command

```
# ansible all -m ping
```

- An ad-hoc command for varying an IBM i device on is shown in Example 3-48.

Example 3-48 Varying IBM i devices using Ansible Ad-Hoc Command

```
# ansible ibmi -m ibm_power_ibmi.ibm_i_device_vary -a 'device_list=OPT02,status=*ON
joblog=false'
```

Ad-hoc commands and playbooks for IBM i

Ad-hoc commands are particularly useful for straightforward tasks, such as variably activating or deactivating resources on IBM i, also testing the reachability of managed hosts. In this process, the Ansible controller dispatches a Python script to the managed nodes. The script reports back the success or failure of the operation. Ad-hoc commands are designed to trigger only one module and its corresponding set of arguments at a time.

On the other hand, playbooks are ideal for more intricate configurations or orchestration scenarios, often involving a series of tasks that collectively execute a larger action using modules. In the context of IBM i modules, common core modules such as **find** are supported. Playbooks are particularly well-suited for various DevOps practices. Some typical scenarios where Playbooks excel include IBM i configuration management, orchestrating IBM i deployments, and managing tasks after deployments have taken place.

Crafting effective YAML playbooks for IBM i

Utilizing Ansible playbooks introduces a powerful approach to orchestrate a multitude of tasks across multiple IBM i systems. These playbooks enable execution of various actions, such as ensuring consistent configurations, performing common tasks, executing deployments, and much more. Expressed in YAML syntax, playbooks provide a structured framework for orchestrating tasks, enhancing efficiency, and promoting system consistency.

- ▶ **Incorporate comments:** To enhance readability and comprehension, comments using the “#” symbol can be seamlessly integrated into your YAML playbook. These comments provide contextual insights and explanations for the included tasks or configurations.
- ▶ **Playbook initialization:** YAML playbooks commence with the declaration of three hyphens (---). This initiates the definition of the playbook's structure and content, creating a clear demarcation for subsequent actions.
- ▶ **Specifying host inventory:** The playbook explicitly designates the host inventory that will be engaged in the orchestrated tasks. This specification ensures that the defined tasks are accurately targeted towards the intended systems.
- ▶ **Gathering facts consideration:** For efficient playbook execution, it is advisable to disable unnecessary fact gathering from managed nodes. In Ansible versions 2.8 and beyond, the directive “gather_facts: no” is employed to curtail fact collection.
- ▶ **Harnessing IBM i collections:** IBM i playbooks benefit from dedicated collections that encapsulate tailored modules and functionalities. By including the relevant collection (for example., “ibm.power_ibmi”) in the playbook, administrators can seamlessly access IBM i-specific capabilities.
- ▶ **Defining tasks:** The heart of the playbook lies within the “tasks” section, where the orchestrated operations are defined. Each task encompasses:
 - A descriptive name (“name”) elucidating the purpose of the task.
 - The module that carries out the action.
 - Corresponding arguments that tailor the task's behavior.
- ▶ **Multiplexing tasks:** The playbook accommodates the definition of multiple tasks, enabling the execution of a series of actions in a coherent sequence. This versatility empowers administrators to create comprehensive automation scenarios.
- ▶ **Concluding the playbook:** The playbook can be concluded using three dots (“...”), signifying the end of the defined content. This conclusion is optional but contributes to maintaining a well-structured playbook

Example 3-49 on page 151 provides a comprehensive playbook sample for IBM i that showcases various YAML syntax features.

Example 3-49 Sample playbook for IBM i

```
# Sample Playbook for IBM i
---
hosts: ibmi
gather_facts: no
collections:
  - ibm.power_ibmi
tasks:
  - name: Display a system value
    ibmi_sysval:
      sysvalue:
        - {'name': 'qccsid'}
      register: dspsysval_ccsid_result

  - name: Display the returned parameters
    debug:
      msg: "{{ chksysval_qmaxsign_result }}"
...

```

Note: By following these guidelines and crafting well-structured YAML playbooks, administrators unlock Ansible's full potential for IBM i. This approach optimizes automation efficacy, ensuring consistency and reliability across IBM i systems. For further insights into YAML syntax, consult the comprehensive YAML documentation at [YAML site](#).

Idempotency and its significance on IBM i

Idempotency serves as a characteristic that amplifies the value of executing Ansible playbooks on IBM i environments. Understanding its implications can greatly enhance the predictability and stability of your operations.

- 1. Reliable execution, consistent results:** Idempotency denotes the remarkable attribute of Ansible playbooks, enabling them to be executed multiple times without altering or disrupting existing configurations. This unique quality ensures that when a playbook is run repeatedly, it consistently produces the same outcomes, contributing to the reliability and predictability of your automation processes.
- 2. Intrinsic to modules:** The concept of Idempotence is intrinsic to Ansible modules, forming a fundamental pillar of their design philosophy. This means that Ansible modules aim to achieve consistent results regardless of how many times they are invoked, reinforcing the reliability of your automation tasks.
- 3. Idempotency on IBM i:** Importantly, Ansible modules on IBM i are inherently idempotent by default. This foundational quality ensures that the operations performed by these modules adhere to the principles of idempotency, simplifying management and minimizing the risk of unintended alterations.
- 4. Practical example:** To illustrate, consider the IBM i module “cmd” sourced from the official repository ([ibmi_cl_command.rst](#)). Let's take the scenario of calling the program **CALL QZLSMAINT PARM('40' '1' '0x100')**. In this case, the configuration flags are cumulative, where repeated executions can add *0x100* to the value. This can potentially disrupt the NetServer configuration. Notably, there is no inherent safeguard to prevent administrators from unintentionally affecting the NetServer configuration through repeated calls to **QZLSMAINT**.

Note: You can ensure consistent and dependable automation outcomes by grasping the principle of idempotency and its implementation within Ansible modules on IBM i. This awareness permits administrators to utilize automation confidently while safeguarding against unintended consequences, ensuring the stability and integrity of your systems.

Utilizing the power_ibmi modules

When working within the IBM i environment, Ansible offers a comprehensive set of modules designed to streamline administrative and deployment tasks. These modules serve as fundamental building blocks, enabling interaction with IBM i systems. With a diverse array of modules available, each is meticulously crafted to perform specific tasks, effectively covering a wide spectrum of administrative and deployment needs.

To harness the power of these modules, it is essential to understand their parameters and data types. Each module is accompanied by its own set of parameters, each tailored to cater to distinct functionalities. These parameters dictate the behavior and configuration of the module during execution. Whether you are creating, modifying, or managing resources on the IBM i, mastering the usage of these parameters ensures efficient and accurate task execution.

For comprehensive guidance on the usage of these modules, referring to the official documentation is crucial. This documentation, accessible at [IBM i modules](#), provides in-depth insights into each module's capabilities, parameter details, and usage examples. By exploring the documentation, IBM i administrators and developers can unlock the full potential of Ansible's **power_ibmi** modules, making way for smoother, more efficient administration and deployment processes.

Executing precise CL commands using the ibmi_c1_command

The **ibmi_c1_command** module plays a crucial role in Ansible's toolkit for IBM i systems, providing administrators and operators with a direct method to execute CL commands accurately. This module's core functionality is centered around the **cmd** parameter, which serves as the conduit for passing a specific CL command for execution.

However, it is important to recognize that the **ibmi_c1_command** module has specific boundaries. Unlike a conventional 5250 emulator, this module does not facilitate interaction with menus or commands in the same way. Instead, its primary purpose is to efficiently execute CL commands in a controlled and automated manner.

A significant feature of this module is its adaptability regarding user context. The module can be configured to run either as the user establishing the SSH connection, usually an administrator, or as a designated user using the **“become_user”** and **“become_user_password”** parameters. This adaptability ensures tasks are executed within the desired user context, accommodating various operational scenarios.

By mastering the **ibmi_c1_command** module, administrators and operators can harness a powerful tool to manage and automate tasks within the IBM i environment. An exploration of the module's practical applications and real-world instances can provide valuable insights into effectively utilizing its capabilities. Example 3-50 shows a sample of the module.

*Example 3-50 Sample of **ibmi_c1_command** module*

```
- hosts: ibmi
  gather_facts: false
  collections:
    - ibm.power_ibmi
  tasks:
    - name: Display the user profile
      ibmi_c1_command:
```

```

cmd: dspusrprf usrprf (AVALOSM)
output: "*PRINT"
register: dspusrprf_result

- name: Print the user profile stdout lines
debug:
  msg: "{{ dspusrprf_result.stdout_lines }}"

```

Discovering IBM i objects with precision using object_find

The **object_find** module emerges as a valuable asset within Ansible's array of tools for IBM i systems, offering an approach to locating specific IBM i objects based on user-defined criteria. This module operates as a versatile search mechanism, capable of honing in on objects by employing various criteria that can be combined using logical "AND" operators.

A standout feature of the **object_find** module is its ability to conduct searches using a comprehensive range of parameters. These parameters permit attributes such as object age, size, name, type, and library, among others. This versatility grants administrators and operators the means to precisely pinpoint objects within the IBM i environment, catering to a multitude of scenarios and requirements.

Behind the scenes, the **object_find** module relies on the integration of IBM i's **QSYS2.OBJECT_STATISTICS** and **QSYS2.SYS_SCHEMAS** views. This integration forms the backbone of the module's efficiency, allowing it to swiftly retrieve pertinent information and present it in a coherent and actionable manner.

By using the **object_find** module, administrators and operators can swiftly navigate and extract valuable insights from their IBM i systems. The module's ability to execute nuanced searches enhances the management and automation of tasks, providing a versatile solution to address diverse operational needs. Example 3-51 shows an example of the module described on this section.

Example 3-51 Sample of object_find module

```

- hosts: ibmi
gather_facts: false
collections:
  - ibm.power_ibmi
tasks:
  - name: Find all journals and journal receivers in library AVALOSM
    ibm.power_ibmi.ibm_object_find:
      object_name: '*ALL'
      object_type_list: '*JRN *JRNRCV'
      lib_name: 'AVALOSM'
      age: '1w'
      age_stamp: 'ctime'
      register: journals

  - name: Print the user profile stdout lines
debug:
  msg: "{{ journals }}"

```

Extracting insights with SQL precision using ibmi_sql_query

The **ibmi_sql_query** module stands as a potent tool within the Ansible toolkit that permits administrators and operators to harness the power of SQL to retrieve valuable information from DB2 for IBM i. Operating as a bridge to the extensive wealth of system information held within DB2, this module streamlines the process of querying data, making it an indispensable asset for effective system management.

One of the core strengths of the `ibmi_sql_query` module is its seamless execution of SQL queries. By interfacing with DB2 for i, users can tap into a treasure trove of insights housed within tables, views, and various SQL Services. These insights span crucial aspects of system functioning, offering a comprehensive view of system health, resource allocation, and performance metrics.

Furthermore, the `ibmi_sql_query` module extends its functionality by allowing users to specify an `expected_row_count` parameter. This added feature permits administrators to fine-tune their querying tasks, enabling them to define expectations for query results. If the actual result set does not meet the defined expectations, the module can be configured to trigger a task failure, providing an automated quality assurance mechanism.

By integrating the capabilities of SQL querying into Ansible workflows, the `ibmi_sql_query` module bolsters the toolkit for IBM i administrators and operators. This module's ability to explore into the inner workings of DB2 for i enhances the precision of system monitoring, diagnostics, and optimization. As a result, Ansible users can navigate the intricacies of their IBM i environments with greater efficiency and depth of insight. Example 3-52 displays a sample using the SQL query module.

Example 3-52 Sample for SQL query

```
- hosts: ibmi
  gather_facts: false
  collections:
    - ibm.power_ibmi
  tasks:
    - name: check if the user already exists
      ibm.power_ibmi.ibmi_sql_query:
        sql: "SELECT * FROM QSYS2.USER_INFO_BASIC WHERE AUTHORIZATION_NAME = 'AVALOSM'"
        register: user_query_result

    - name: display the result from the query
      debug:
        msg: "{{ user_query_result }}"

    - name: assert the user doesn't exist
      assert:
        that: (user_query_result.row | length) == 0
```

3.4.4 VIOS as Ansible-managed client

Businesses are turning to PowerVM virtualization to consolidate multiple workloads onto fewer systems to increase server use, and to reduce cost. PowerVM provides a secure and scalable virtualization environment for AIX, Linux, and IBM i applications that are built on the advanced reliability, availability, and serviceability features and the leading performance of the IBM Power systems platform.

The Virtual I/O Server (VIOS) is part of the PowerVM hardware feature. The VIOS is software that is located in a logical partition running in your IBM Power server that provides virtualization functionality for the other LPARs running in that server. The VIOS provides virtual Small Computer Serial Interface (SCSI) target, virtual Fibre Channel, Shared Ethernet Adapter, and PowerVM Active Memory Sharing capability to client logical partitions within the system. The VIOS also provides the Suspend/Resume feature to AIX, IBM i, and Linux client logical partitions within the system. You can use the VIOS to perform the following functions:

- Sharing of physical resources between logical partitions on the system
- Creating logical partitions without requiring additional physical I/O resources

- Creating more logical partitions than there are I/O slots or physical devices available with the ability for logical partitions to have dedicated I/O, virtual I/O, or both
- Maximizing use of physical resources on the system
- Helping to reduce the storage area network (SAN) infrastructure

The Virtual I/O Server is configured and managed through a command-line interface. All aspects of Virtual I/O Server administration can be accomplished through the command-line interface, including the following:

- Device management (physical, virtual, logical volume manager (LVM))
- Network configuration
- Software installation and update
- Security
- User management
- Maintenance tasks

Setting up your VIOS server for Ansible client

The default user for connecting to a VIOS LPAR is *padmin*, which has access to the *ioscli* shell. We have chosen to create a unique user for our Ansible management environment and in this scenario we will create a user *ansible* for our connection. This is shown in Example 3-53.

Example 3-53 Create ansible userid in the VIOS

```
Last unsuccessful login: Wed Nov 10 12:23:52 CST 2021 on ssh from sltsmnim
Last login: Wed May 24 07:42:54 CDT 2023 on /dev/pts/1 from 192.168.184.201
The most recent software update has modified the current system rules.
These modifications have not been deployed on the system. To view the
modifications and deploy, run the 'rulescfgset' command.
$ oem_setup_env
# mkuser roles=PAdmin,CacheAdm,FSAdmin,pkgadm \
    default_roles=PAdmin,CacheAdm,FSAdmin,pkgadm ansible>
# id ansible
uid=205(ansible) gid=1(staff)
# pwdadm -c ansible
```

Starting with VIOS 4.1.0.10, python is included in the VIOS image. If you are using VIOS levels earlier than that you will need to set up python. Example 3-54 shows that a simple Ansible command is unsuccessful because python is not natively installed on the VIOS prior to version 4.1.0.10.

Example 3-54 Ansible command failure due to Ansible not being set up

```
[root@ansible-AAP-redbook playbooks]# cat facts.yml
---
- hosts: all
  remote_user: ansible

  tasks:
  - name: print facts
    debug:
      var: ansible_facts
[root@ansible-AAP-redbook playbooks]# ansible-playbook -i vios_inventory.yml facts.yml
--ask-pass
SSH password:

PLAY [all] *****
TASK [Gathering Facts] *****
```

```
fatal: [vio]: FAILED! => {"ansible_facts": {}, "changed": false, "failed_modules":
{"ansible_legacy_setup": {"failed": true, "module_stderr": "Shared connection to
bergessiovios1 closed.\r\n", "module_stdout": "/bin/sh: /usr/bin/python: not found.\r\n",
"msg": "MODULE FAILURE\nSee stdout/stderr for the exact error", "rc": 127}}, "msg": "The
following modules failed to execute: ansible_legacy_setup\n"}
```

```
PLAY RECAP *****
vio                : ok=0    changed=0    unreachable=0    failed=1    skipped=0
rescued=0    ignored=0
```

```
[root@ansible-AAP-redbook playbooks]
```

At this point you can one of the following:

1. Install dnf from the AIX toolbox
2. Install dnf using the power of Ansible to run the install

We opted to do option 2 and we created the playbook `dnf-vios.yml` to do the install. This is shown in Example 3-55.

Example 3-55 Setup su for ansible user, and vios-dnf.yml

```
$ oem_setup_env
# passwd root
Changing password for "root"
root's New password:
Enter the new password again:
# pwdadm -c root
# chuser su=true root

# cat -n dnf-vios.yml

- hosts: all
  remote_user: ansible
  gather_facts: no
  become: true
  become_method: su

  tasks:
    - name: download dnf_aixtoolbox.sh
      get_url:
        url: https://public.dhe.ibm.com/aix/freeSoftware/aixtoolbox/eziinstall/ppc/dnf_aixtoolbox.sh
        dest: /tmp/dnf_aixtoolbox.sh
        mode: 0755
        delegate_to: localhost
    - name: copy dnf_aixtoolbox.sh to vios
      raw: "scp -p /tmp/dnf_aixtoolbox.sh {{ ansible_user }}@{{ inventory_hostname
}}:/tmp/dnf_aixtoolbox.sh
        delegate_to: localhost
    - name: execute dnf_aixtoolbox.sh on vios
      raw: "chmod 755 /tmp/dnf_aixtoolbox.sh ; /tmp/dnf_aixtoolbox.sh -y"

# ansible-playbook -K -i viol, dnf-vios.yml
BECOME password:

PLAY [all]
*****
****
```

```

TASK [download dnf_aitoolbox.sh]
*****
ok: [viol - localhost]

TASK [copy dnf_aitoolbox.sh to vios]
*****

changed: [viol - localhost]
*****

TASK [update virtual RPM packages]
*****

changed: [viol]

TASK [execute dnf_aitoolbox.sh on
vios]*****

changed: [viol]

PLAY RECAP
*****
****

viol          :oks4 changed=3 unreachable=@ failed=0 skipped=0 rescued=0
ignored=0

```

Now we have dnf installed in our VIOS which is required to install python.

There will be multiple VIOS in your environment as it is suggested for high availability that there be at least two VIOS per server frame – acting as an active/active failover solution. If you have additional requirements for separation of environments for security, there may be additional VIOS on your server and there will almost certainly be multiple servers in your environment so there will be many VIOS to manage.

We need to be sure we can access all of the VIOS LPARs. We can manually create the user on each VIOS, or we just can use the power of Ansible. We chose to do it with Ansible and we created a shell script as shown in Example 3-56 for this purpose.

Example 3-56 Script to create users on VIOS servers

```

#!/usr/bin/expect -f

if { $argc != 3 } {
send_error "Usage : $argv@ ssh-connection ssh-password user-password\n"
exit 1
}

log_user 1

set timeout 60

set sshconn [lindex $argv 0]
set sshpw [lindex $argv 1]
set newpw [Llindex $argv 2]

spawn ssh $sshconn

```

```

expect "password: "
send "$sshpw\r"

expect "§ "

send "oem_setup_env\r"

expect "# "

send "mkuser roles=PAdmin,CacheAdm,FSAdmin,pkgadm,SysBoot,isso
default_roles=PAdmin,CacheAdm,FSAdmin,pkgadm,SysBoot,isso ansible\r"
expect "# "

send "chuser su=true root\r"

expect "# "

send "echo 'ansible:$newpw' | chpasswd -c\r"
expect "# "

send "echo 'root:$newpw' | chpasswd -c\r"
expect "# "

send "exit\r"
send "exit\r"

```

The script shown above, will be used as part of our playbook to run the user creation on every VIOS Server.

We have the script, now we'll build an inventory to run that playbook as shown in Example 3-57.

Example 3-57 Inventory build for Ansible user creation playbook

```

#cat viosinventory
[vios]
vios1
vios2
vios3
vios4
[all:vars]
ansible_connection=ssh
ansible_user=admin
ansible_password=adminpassword

```

The playbook we used create the *ansible* user on each VIOS is shown in Example 3-58.

Example 3-58 Playbook create-vios-user.yml

```

#cat create-vios-user.yml
---
- name: create remote user
  raw: "{{ download_dir }}/user.e {{ ansible_user }}@{{ inventory_hostname }} {{
ansible_password }} {{ new_password }}"
  delegate_to: localhost

- name: copy ssh public key
  raw: "SSHPASS={{ new_password }} sshpass -e ssh-copy-id ansible@{{ inventory_hostname }}"
  delegate_to: localhost

```

```
#ansible-playbook -i vios-inventory create-vios-user.yml
```

So we have our users, created we have our inventory we can install python on all the partitions, with a similar script than the one we used for creating the users. This is shown in Example 3-59.

Example 3-59 Install python3 and pip3 script

```
#!/usr/bin/expect -f

if { $argc != 3 } {
send_error "Usage : $argv@ ssh-connection ssh-password user-password\n"
exit 1
}

log_user 1

set timeout 60

set sshconn [lindex $argv 0]
set sshpw [lindex $argv 1]

spawn ssh $sshconn
expect "password: "
send "$sshpw\r"

expect "$ "

send "oem_setup_env\r"

expect "# "

send "dnf install python3 pip3 -y \r"
expect "# "
```

Now that our VIOS clients are ready for use, we'll walk through a set of use cases for the VIOS collection. These include:

- VIOS upgrade
- VIOS backup
- VIOS mapping
- VIOS hardening

First we need to validate that our collection is installed and ready. In Example 3-60 we validate the versions of the collections that are installed.

Example 3-60 Validating ibm.power collections

```
root@ansible-AAP-redbook ~]# ansible-galaxy collection list
[WARNING]: Collection at '/root/.ansible/collections/ansible_collections/ibm/aix' does not have a
MANIFEST.json file, nor has it galaxy.yml: cannot detect version.

# /root/.ansible/collections/ansible_collections
Collection      Version
-----
community.general 6.6.0
ibm.aix          *
ibm.power_aix    1.6.4
ibm.power_hmc    1.8.0
ibm.power_vios   1.2.3
```

```
[root@ansible-AAP-redbook ~]#
```

VIOS upgrade

With our collections installed and ready, we'll start using them to write some powerful Ansible scripts for our servers. Example 3-61 shows a playbook used to update the VIOS software.

Example 3-61 vios-update.yml playbook

```
#cat vios-update.yml

- hosts: all
  remote_user: ansible
  gather_facts: no
  collections:
  - ibm.power_aix
  - ibm.power_vios

  roles:

  - name: Bootstrap VIOS
    role: bootstrap_vios
    become: true
    become_method: su

  - name: Update VIOS (1st part)
    role: update_vios
    when: "'v1' in inventory_hostname"

  - name: Update VIOS (2nd part)
    role: update_vios
    when: "'v2' in inventory_hostname"
```

The magic behind the playbook, is in the roles. Example shows details on the update_vios role.

Example 3-62 Display role update_vios

```
#cat roles/update_vios/tasks/main.yml

- name: commit all uncommitted updates
  ibm.power_vios.updateios:
    action: commit

- name: mount remote repository with updates
  ibm.power_aix.mount:
    state: mount
    node: "{{ repo_node }}"
    mount_dir: "{{ repo_dir }}"
    mount_over_dir: "{{ local_dir }}"

- name: update VIO server
  ibm.power_vios.updateios:
    action: update
    device: "{{ local_dir }}"
    accept_licenses: yes
```

However, the vios-update playbook shown in Example 3-61 is not complete as it should handle a couple more issues:

- Commit can fail with RC = 19 or 20 which means “Already committed”.
- After update you should reboot your VIOS.

You just can manually execute the *updateios* playbook by on the command line and ignore some return codes as shown in Example 3-63.

Example 3-63 Code to commit and ignore return codes 19 and 20

```
---
- name: commit all uncommitted updates
  shell:
  cmd: fusr/ios/cli/ioscli updateios -commit
  register: result
  failed_when: ( result.rc not in [ 0, 19, 20 ] )
```

Backup VIOS

Now let us look at some playbooks to backup your VIOS. These will run *viosbr* which basically is an *mksysb* for VIOS Servers. This Ansible playbook shown in Example 3-64 will execute *viosbr* and direct the output to a NFS share location.

Example 3-64 Playbook to run viosbr

```
#cat viosbr-playbook.yml
---
- name: backup vios
  hosts: all
  gather_facts: false
  remote_user: ansible
  vars:
  ansible_python_interpreter: /opt/freeware/bin/python3

  tasks:
  - name: get current date
    ansible.builtin.shell: "date +%Y%m%d"
    register: ourdate
    delegate_to: localhost

  - name: create viosbr backup
    ibm.power_vios.vioshr:
      action: backup
      file: "/home/ansible/vioshr.{{ ourdate.stdout }}"

  - name: mount NFS share for backups
    ibm.power_aix.mount:
      state: mount
      node: nim
      mount_dir: /backup
      mount_over_dir: /mnt

  - name: create mksysb of VIO
    ibm.power_vios.backupios:
      file: "/mnt/{{ inventory_hostname }}_mksysb.{{ ourdate.stdout }}"
      mksysb: true

  - name: unmount NFS share
    ibm.power_aix.mount:
      state: umount
      mount_over_dir: /mnt
```

VIOS mapping

We can use an Ansible playbook to get facts about our VIOS Servers like with the *vios-facts.yml*, script bellow

Example 3-65 vios-facts.yml

```
#cat vios-facts.yml
---
- name: get VIO mapping
  hosts: all
  remote_user: ansible
  gather_facts: false
  vars:
    ansible_python_interpreter: /opt/freeware/bin/python3

  tasks:
    - name: get mapping facts
      ibm.power_vios.mapping_facts:
    - name: print facts
      ansible.builtin.debug:
        var: ansible_facts
```

VIOS hardening

Out last example script, Example 3-66, is written to set the security level for the vios.

Example 3-66 vios-security.yml

```
#cat vios-security.yml
---
- name: apply secure configuration
  ibm.power_vios.viosecurer:
    level: low
```

As you saw during this chapter, there is a lot you can do with the VIOS collection, using a mix of the HMC, AIX, and VIOS collections.

3.4.5 Red Hat OpenShift as Ansible-managed client

Red Hat OpenShift on Power can run in many different environments. For example You can use a bare metal installation on your Power Server or you can install on LPARs which have been prebuilt using PowerVM, These two scenarios are mostly manual and they do not gain much advantage from the use of Ansible (although you might have used Ansible to build the LPARs).

Ansible will be more help if you are deploying your OpenShift cluster on PowerVS, on PowerVM managed by PowerVC, or on Power servers managed by KVM. These are the scenarios we will describe in this section. The environment is shown in Figure 3-22 on page 163.

Installing OpenShift on PowerVS using Ansible

PowerVS features a Cli for managing and creating your servers, In this section we will show you scripts that use Terraform and Ansible to deploy a Red Hat OpenShift cluster on PowerVS using ocp4-upi-powervs for the deployment process.

You must have an IBM Cloud account to deploy PowerVS resources. To ensure that your PowerVS instance is capable of deploying OpenShift clusters, make sure that your account has the proper permissions and validate that you have created the appropriate security certificates.

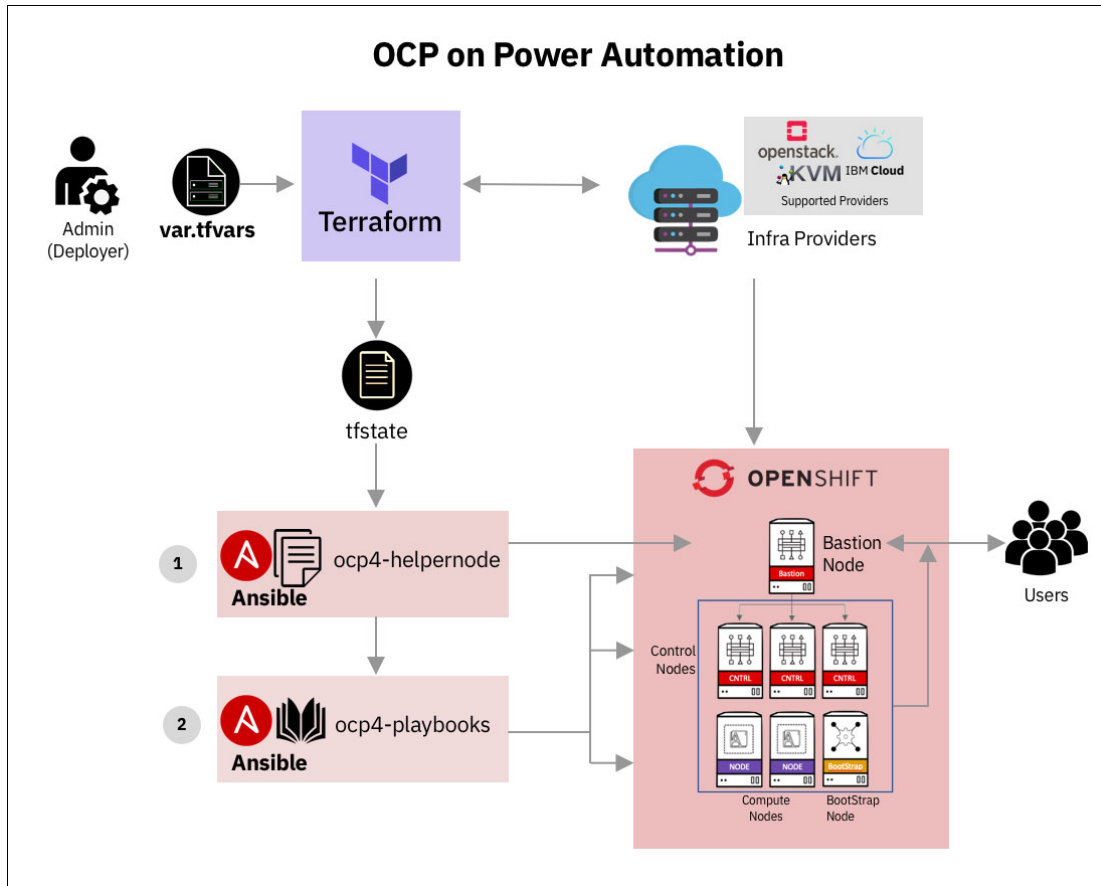


Figure 3-22 OpenShift deploy with Ansible Scenarios

Detailed prerequisites

This section helps you ensure that your IBM Cloud account is set up and that you will be able to create the PowerVS instance for your OpenShift cluster. The following steps will help you set up your environment:

1. Validate that you have an IBM Cloud account to create your Power Systems Virtual Server instance. If you don't already have one, you can create an account at: <https://cloud.ibm.com>.
2. Create an IBM Cloud account API key. For information on setting up your API key reference *IBM Power Virtual Server Guide for IBM AIX and Linux*, SG24-8512 or see the [IBM Cloud documentation](#).
3. After you have an active IBM Cloud account, you can create a Power Systems Virtual Server service, you can reference *IBM Power Virtual Server Guide for IBM AIX and Linux*, SG24-8512 for more details.
4. Next, request an OpenShift Pull secret. Download the secret from <https://cloud.redhat.com/openshift/install/power/user-provisioned>. You'll need to place the file in the `install` directory and name it `pull-secret.txt`. You will need a RHEL Subscription ID and Password.

Getting ready for installation

The install will use a simple script based installer which will deploy an OpenShift cluster on PowerVS. The script can be run in multiple platforms including Linux(x86_64/ppc64le), Windows & Mac OSX.

The PowerVS instance will require some special network permissions to ensure inbound access is allowed for TCP ports for ssh (22) and to allow outbound access for http (80), https (443), and OC CLI (6443). This is only required when using a Cloud instance or a remote VM so that you can connect to it using SSH and run the installer.

1. Create an install directory where all the configurations, logs and data files will be stored.

```
[root@ansible-AAP-redbook ~]# mkdir ocp-install-dir && cd ocp-install-dir
```

2. Download the script on your system and change the permission to execute as shown in Example 3-67.

Example 3-67 Change script to allow execution

```
[root@ansible-AAP-redbook ocp-install-dir]# curl -sL
https://raw.githubusercontent.com/ocp-power-automation/openshift-install-power/main/openshi
ft-install-powervs -o ./openshift-install-powervs
[root@ansible-AAP-redbook ocp-install-dir]# ls -lrt
total 64
-rw-r--r--. 1 root root 62001 Sep 29 15:24 openshift-install-powervs
[root@ansible-AAP-redbook ocp-install-dir]# chmod +x ./openshift-install-powervs
[root@ansible-AAP-redbook ocp-install-dir]#
```

3. Running the script without parameters displays the help information for the script as shown in Example 3-68.

Example 3-68 Help information for the script

```
[root@ansible-AAP-redbook ocp-install-dir]# ./openshift-install-powervs
```

Automation for deploying OpenShift 4.X on PowerVS

Usage:

```
openshift-install-powervs [command] [<args> [<value>]]
```

Available commands:

setup	Install all the required packages/binaries in current directory
variables	Interactive way to populate the variables file
create	Create an OpenShift cluster
destroy	Destroy an OpenShift cluster
output	Display the cluster information. Runs terraform output [NAME]
access-info	Display the access information of installed OpenShift cluster
help	Display this information

Where <args>:

-var	Terraform variable to be passed to the create/destroy command
-var-file	Terraform variable file name in current directory. (By default using var.tfvars)
-flavor	Cluster compute template to use eg: small, medium, large
-force-destroy	Not ask for confirmation during destroy command
-ignore-os-checks	Ignore operating system related checks
-ignore-warnings	Warning messages will not be displayed. Should be specified first, before any other args.
-verbose	Enable verbose for terraform console messages
-all-images	List all the images available during variables prompt
-trace	Enable tracing of all executed commands
-version, -v	Display the script version

Environment Variables:

IBMCLOUD_API_KEY	IBM Cloud API key
RELEASE_VER	OpenShift release version (Default: 4.13)
ARTIFACTS_VERSION	Tag or Branch name of ocp4-upi-powervs repository (Default: main)

RHEL_SUBS_PASSWORD RHEL subscription password if not provided in variables
 NO_OF_RETRY Number of retries/attempts to run repeatable actions such as create
 (Default: 5)

Submit issues at: <https://github.com/ocp-power-automation/openshift-install-power/issues>

[root@ansible-AAP-redbook ocp-install-dir]#

4. Set up your environment by exporting the IBM Cloud API Key and RHEL Subscription Password as shown in Example 3-69.

Example 3-69 Set environment variables

```
$ set +o history
$ export IBM_CLOUD_API_KEY='<your API key>'
$ export RHEL_SUBS_PASSWORD='<your RHEL subscription password>'
$ set -o history
```

5. Run the create command: **\$./openshift-install-powervs create**

The script will setup the required tools and run in interactive mode prompting for inputs.

6. Once the above command completes successfully, it will print the cluster access information. Login to bastion:

```
'ssh -i automation/data/id_rsa root@XXX.XXX.XXX.XXX'
```

Now start using the 'oc' command.

To access the cluster on local system when using 'oc' run:

```
'export KUBECONFIG=/root/ocp-install-dir/automation/kubeconfig'
```

7. Access the OpenShift web-console here:

```
https://console-openshift-console.apps.test-ocp-6f2c.ibm.com
```

8. Login to the console with

```
user: "kubeadmin", and password: "MHvmI-z5nY8-CBKFH-hmCDJ"
```

9. In order to access your brand new cluster without defining a dns for your OpenShift Cluster, add these lines to your local system 'hosts' file:

```
XXX.XXX.XXX.XXX.,api.test-ocp-6f2c.ibm.com
console-openshift-console.apps.test-ocp-6f2c.ibm.com
integrated-oauth-server-openshift-authentication.apps.test-ocp-6f2c.ibm.c
om oauth-openshift.apps.test-ocp-6f2c.ibm.com
prometheus-k8s-openshift-monitoring.apps.test-ocp-6f2c.ibm.com
grafana-openshift-monitoring.apps.test-ocp-6f2c.ibm.com
bolsilludo.apps.test-ocp-6f2c.ibm.com
```

Advanced Usage

Before running the script, you may choose to override some environment variables depending on your requirements. By default OpenShift version 4.12 is installed. If you want to install 4.11, then export the following variables:

```
$ export RELEASE_VER="4.11"
ARTIFACTS_VERSION: Tag/Branch (eg: release-4.11, v4.11, main) of
ocp4-upi-powervs repository. Default is "main".
$ export ARTIFACTS_VERSION="release-4.11"
```

Non-interactive mode

You can avoid the interactive mode by having the required input files available in the install directory. The required input files are:

1. SSH key files (filename: id_rsa & id_rsa.pub)
2. Terraform vars file (filename: var.tfvars). An example is shown in Example 3-70.

Example 3-70 Example `var.tfvars` file

```

---
ibmcloud_region = "syd"
ibmcloud_zone = "syd04"
service_instance_id = "123456abc-xzz-2223434343"
rhel_image_name = "rhel-83-12062022"
rhcos_image_name = "rhcos-412-02012023"
network_name = "ocp-net"
openshift_install_tarball =
"https://mirror.openshift.com/pub/openshift-v4/ppc64le/clients/ocp/stable-4.12/openshift-in
stall-linux.tar.gz"
openshift_client_tarball =
"https://mirror.openshift.com/pub/openshift-v4/ppc64le/clients/ocp/stable-4.12/openshift-cl
ient-linux.tar.gz"
cluster_id_prefix = "test-ocp"
cluster_domain = "xip.io"
storage_type = "nfs"
volume_size = "300"
bastion = {memory = "16", processors = "1", "count" = 1}
bootstrap = {memory = "32", processors = "0.5", "count" = 1}
master = {memory = "32", processors = "0.5", "count" = 3}
worker = {memory = "32", processors = "0.5", "count" = 2}
rhel_subscription_username = "mysubscription@email.com"
rhel_subscription_password = "mysubscriptionPassword"
---

```

You can also pass a custom Terraform variables file using the option `-var-file <filename>` to the script. You can also use the option `-var "key=value"` to pass a single variable. If the same variable is given more than once then precedence will be from left (low) to right (high).

For a flowchart of how the script and process works, please take a look at Figure 3-23 on page 167.

Deploying an OpenShift Cluster using PowerVC

This section provides a methodology for installing a Red Hat OpenShift cluster on your Power infrastructure managed by IBM PowerVC. The process is similar to the one used to install OpenShift on PowerVS as described in “Installing OpenShift on PowerVS using Ansible” on page 162 and also uses Terraform. Terraform version 1.2.0 and above is required.

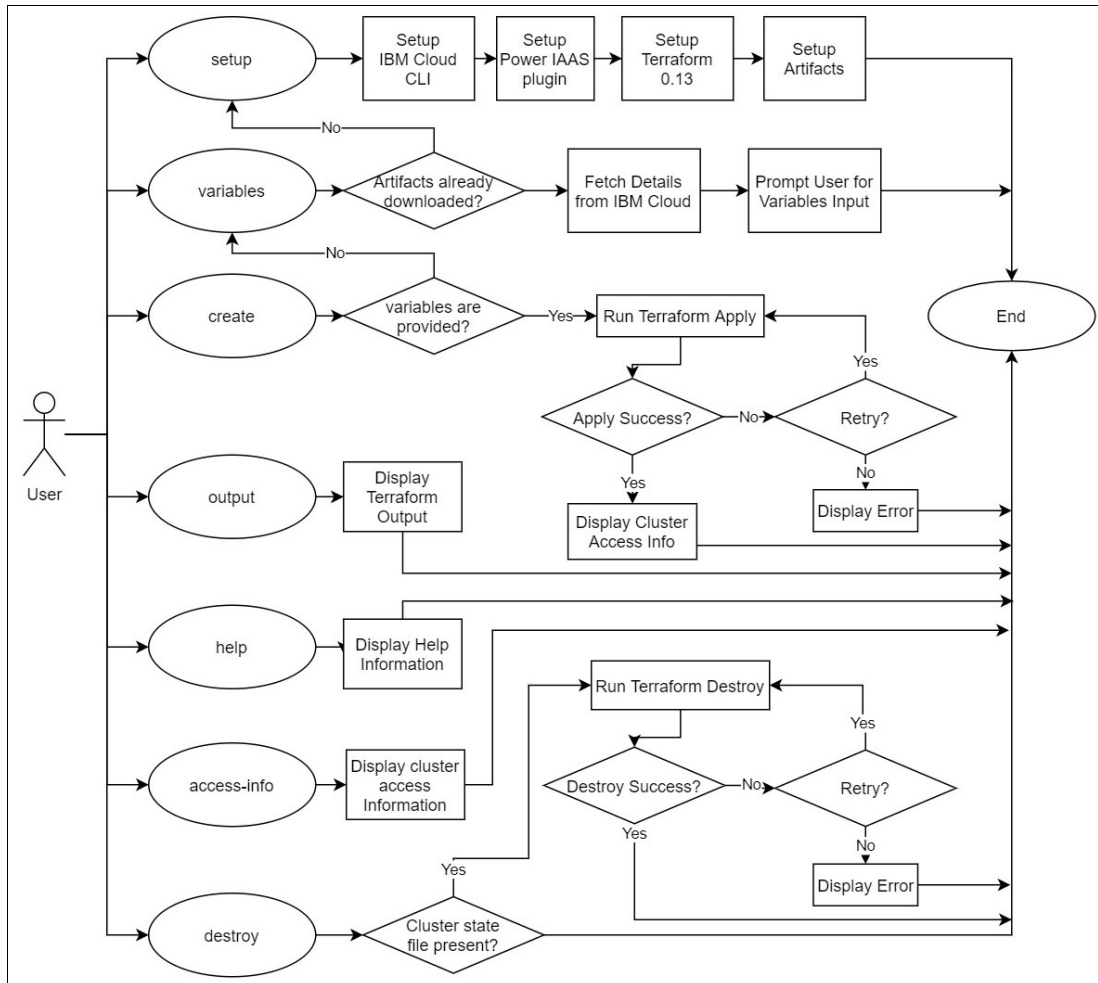


Figure 3-23 Graphical Description of the openshift-install-powervs script interactions

Install Terraform and providers for Power environment

Before starting to install your OpenShift cluster using this process you need to first install Terraform for your Power environment. To do so:

1. Download and install the latest Terraform binary for Linux/ppc64le from <https://github.com/ppc64le-development/terraform-ppc64le/releases>.
2. Download the required Terraform providers for Power into your TF project directory as shown in Example 3-71.

Example 3-71 Download and install Terraform

```

$ cd <path_to_TF_project>
$ mkdir -p ./providers
$ curl -fsSL https://github.com/ocp-power-automation/terraform-providers-power/releases/download/v0.11/archive.zip -o archive.zip
$ unzip -o ./archive.zip -d ./providers
$ rm -f ./archive.zip
Initialize Terraform at your TF project directory:
$ terraform init --plugin-dir ./providers
  
```

The Ansible code for this process is provided by the [ocp4-upi-powervm project](#) and it provides Terraform based automation code to help the deployment of OpenShift Container Platform (OCP) 4.x on PowerVM systems managed by PowerVC. This project leverages the same Ansible playbook internally for OCP deployment on PowerVM LPARs managed via PowerVC.

If you are not using PowerVC, but instead are using standalone PowerVM LPAR management, then [this guide](#) walks you through the process of using the Ansible playbook to setup a helper node (bastion) to simplify the OCP deployment.

PowerVC Prerequisites

As part of the install process, you will need to create a Red Hat CoreOS (RHCOS) image and a RHEL 8.2 (or later) image in PowerVC. The RHEL 8.x image will be installed on the bastion node and the RHCOS image is installed on the bootstrap, master and worker nodes.

- For RHCOS image creation, follow the steps documented in this [document](#).
- For RHEL image creation follow the steps mentioned in this [document](#). You may either create a new image from ISO or use a similar method to that used for the RHCOS option.

Compute Templates

You'll need to create compute templates for bastion, bootstrap, master and worker nodes. The recommended LPAR configurations are:

Bootstrap:	2 vCPUs, 16GB RAM, 120 GB Disk
Master:	2 vCPUs, 32GB RAM, 120 GB Disk
Worker	2 vCPUs, 32GB RAM, 120 GB Disk
Bastion	2 vCPUs, 16GB RAM, 200 GB Disk

Increase worker and bastion settings based on application requirements.

PowerVM LPARs by default use SMT8, so with 2 vCPUs the number of logical CPUs as seen by the Operating System will be 16 (2 vCPUs x 8 SMT)

Download the Automation Code

You'll need to use *git* to clone the deployment code when working off the master branch using these commands:

```
$ git clone https://github.com/ocp-power-automation/ocp4-upi-powervm.git
$ cd ocp4_upi_powervm
```

All further instructions provide here, assume you are in the code directory:

```
ocp4-upi-powervm
```

Setup Terraform Variables

Update the `var.tfvars` file based on your environment. A description of the variables is available [at this link](#). You can use environment variables for sensitive data that should not be saved to disk as shown in Example 3-72.

Example 3-72 Example of setting environment variables

```
$ set +o history
$ export POWERVC_USERNAME=xxxxxxxxxxxxxxxx
$ export POWERVC_PASSWORD=xxxxxxxxxxxxxxxx
$ export RHEL_SUBS_USERNAME=xxxxxxxxxxxxxxxx
$ export RHEL_SUBS_PASSWORD=xxxxxxxxxxxxxxxx
$ set -o history
```

Start Install

Run the following commands from within the directory:

```
$ terraform init
$ terraform apply -var-file var.tfvars
```

If using environment variables for sensitive data, then do the following, instead.

```
$ terraform init
$ terraform apply -var-file var.tfvars -var user_name="$POWERVC_USERNAME" -var
password="$POWERVC_PASSWORD" -var
rhel_subscription_username="$RHEL_SUBS_USERNAME" -var
rhel_subscription_password="$RHEL_SUBS_PASSWORD"
```

Now wait for the installation to complete. It may take around 40 minutes to complete provisioning.

On successful install cluster details will be printed as shown in Example 3-73.

Example 3-73 Output from Terraform install

```
bastion_private_ip = 192.168.25.171
bastion_public_ip = 16.20.34.5
bastion_ssh_command = ssh -i data/id_rsa root@16.20.34.5
bootstrap_ip = 192.168.25.182
cluster_authentication_details = Cluster authentication details are available in 16.20.34.5
under ~/openstack-upi/auth
cluster_id = test-cluster-9a4f
etc_hosts_entries =
16.20.34.5 api.test-cluster-9a4f.mydomain.com
console-openshift-console.apps.test-cluster-9a4f.mydomain.com
integrated-oauth-server-openshift-authentication.apps.test-cluster-9a4f.mydomain.com
oauth-openshift.apps.test-cluster-9a4f.mydomain.com
prometheus-k8s-openshift-monitoring.apps.test-cluster-9a4f.mydomain.com
grafana-openshift-monitoring.apps.test-cluster-9a4f.mydomain.com
bolsilludo.apps.test-cluster-9a4f.mydomain.com

install_status = COMPLETED
master_ips = [
  "192.168.25.147",
  "192.168.25.176",
]
oc_server_url = https://test-cluster-9a4f.mydomain.com:6443
storageclass_name = nfs-storage-provisioner
web_console_url = https://console-openshift-console.apps.test-cluster-9a4f.mydomain.com
worker_ips = [
  "192.168.25.220",
  "192.168.25.134",
]
```

If you are using a wild card domain name like nip.io or xip.io then *etc_host_entries* is empty as shown in Example 3-74.

Example 3-74 Output when using wild card domain name

```
bastion_private_ip = 192.168.25.171
bastion_public_ip = 16.20.34.5
bastion_ssh_command = ssh -i data/id_rsa root@16.20.34.5
bootstrap_ip = 192.168.25.182
cluster_authentication_details = Cluster authentication details are available in 16.20.34.5
under ~/openstack-upi/auth
```

```

cluster_id = test-cluster-9a4f
etc_hosts_entries =
install_status = COMPLETED
master_ips = [
  "192.168.25.147",
  "192.168.25.176",
]
oc_server_url = https://test-cluster-9a4f.16.20.34.5.nip.io:6443
storageclass_name = nfs-storage-provisioner
web_console_url =
https://console-openshift-console.apps.test-cluster-9a4f.16.20.34.5.nip.io
worker_ips = [
  "192.168.25.220",
  "192.168.25.134",
]

```

This information can be retrieved anytime by running the following command from the root folder of the code:

```
$ terraform output
```

In case of any errors, you'll have to re-apply. Please refer to [known issues](#) to get more details on potential issues and workarounds.

Post Install

Once the deployment is completed successfully, you can safely delete the bootstrap node. This step is optional but recommended so as to free up the resources used. To delete the bootstrap node change the count value to 0 in *bootstrap map* variable and re-run the apply command.

Create API and Ingress DNS Records

Please skip this section if your cluster_domain is one of the online wildcard DNS domains: nip.io, xip.io and sslip.io. For all other domains, you can use one of the following options.

- ▶ Add entries to your DNS server using the general format shown below:

```

api.<cluster_id>. IN A <bastion_public_ip>
*.apps.<cluster_id>. IN A <bastion_public_ip>

```

You'll need the *bastion_public_ip* and *cluster_id* – which were printed at the end of your successful install or you can retrieve those values anytime by running **terraform output** from the install directory.

- ▶ Add entries to your client system *hosts* file

Tip: For Linux and Mac hosts file is located at `/etc/hosts` and for Windows it's located at `c:\Windows\System32\Drivers\etc\hosts`.

The general format is shown in Example 3-75. The entries for your installation were printed at the end of your successful install. Alternatively you can retrieve it anytime by running **terraform output** from the install directory. Append these values to the *host* file.

Example 3-75 Entries for hosts file

```

<bastion_public_ip> api.<cluster_id>
<bastion_public_ip> console-openshift-console.apps.<cluster_id>
<bastion_public_ip> integrated-oauth-server-openshift-authentication.apps.<cluster_id>
<bastion_public_ip> oauth-openshift.apps.<cluster_id>
<bastion_public_ip> prometheus-k8s-openshift-monitoring.apps.<cluster_id>
<bastion_public_ip> grafana-openshift-monitoring.apps.<cluster_id>

```



```
<bastion_public_ip> <app name>.apps.<cluster_id>
```

Cluster Access

Once your cluster is up and running, you can login to the cluster using the OpenShift login credentials in the bastion host. The location will be printed at the end of a successful install, or you can retrieve it anytime by running **terraform output** from the install directory. An example is shown in Example 3-76.

Example 3-76 Login credentials for the OpenShift cluster

```
bastion_public_ip = 16.20.34.5
bastion_ssh_command = ssh -i data/id_rsa root@16.20.34.5
cluster_authentication_details = Cluster authentication details are available in 16.20.34.5
under ~/openstack-upi/auth
```

There are two files under `~/openstack-upi/auth` -

- kubeconfig: can be used for CLI access -
- kubeadmin-password: Password for kubeadmin user which can be used for CLI and UI access

Note: Ensure you securely store the OpenShift cluster access credentials. If desired delete the access details from the bastion node after securely storing them elsewhere.

You can copy the access details to your local system, or a secure password vault:

```
$ scp -r -i data/id_rsa root@158.175.161.118:~/openstack-upi/auth/*
```

Clean up

If you are finished with your cluster and want to destroy it, by running:

```
terraform destroy -var-file var.tfvars
```

This ensures that all resources are properly cleaned up. Do not manually clean up your environment unless both of the following are true:

- You know what you are doing.
- Something went wrong with an automated deletion.

Deploying an OpenShift Cluster on a Power Based KVM Environment

The [ocp4-upi-kvm project](#) provides Terraform based automation code to help the deployment of OpenShift Container Platform (OCP) 4.x on KVM VMs using *libvirt*. This project leverages the previously mentioned Ansible playbook ([ocp4-helpernode](#)) to setup a helper node (bastion) for OCP deployment. The Ansible Script will create a helper node for running the install documented at <https://github.com/redhat-cop/ocp4-helpernode>.

Automation Host Prerequisites

The automation needs to run from a system with internet access. This could be your laptop or a VM with public internet connectivity. This automation code has been tested on the following 64-bit Operating Systems:

- Linux (preferred)
- Mac OSX (Darwin)

The automation host installation is documented in [GitHub ocp-power](#).

This involves Terraform install and creating install images for RHCOS and RHEL.

LibVirt Prerequisites

KVM virtualization relies on *libvirt* to work, so as prerequisites for installing OpenShift on KVM, you'll need to follow the steps detailed at:

<https://ocp-power-automation.github.io/ocp4-upi-kvm/docs/libvirt-host-setup/>

Download the Automation Code

You'll need to use `git` to clone the deployment code when working off the master branch:

```
git clone https://github.com/ocp-power-automation/ocp4-upi-kvm.git
cd ocp4_upi_kvm
```

All further instructions assume you are in the code directory *ocp4-upi-kvm*.

Setup Terraform Variables

Update the *var.tfvars* based on your environment. Description of the variables are available in the following link. You can use environment variables for sensitive data that should not be saved to disk as shown in Example 3-77.

Example 3-77 Setting environment variables for sensitive data

```
$ export RHEL_SUBS_USERNAME=xxxxxxxxxxxxxxxx
$ export RHEL_SUBS_PASSWORD=xxxxxxxxxxxxxxxx
$ set -o history
$ set +o history
```

Start Install

Run the following commands from within the directory.

```
$ terraform init
$ terraform apply -var-file var.tfvars
```

If using environment variables for sensitive data, then do the following, instead.

```
$ terraform init
$ terraform apply -var-file var.tfvars -var
rhel_subscription_username="$RHEL_SUBS_USERNAME" -var
rhel_subscription_password="$RHEL_SUBS_PASSWORD"
```

Now wait for the installation to complete. It may take around 40 minutes to complete provisioning.

On successful install cluster details will be printed as shown in Example 3-78.

Example 3-78

```
bastion_ip = 192.168.61.2
bastion_ssh_command = ssh root@192.168.61.2
bootstrap_ip = 192.168.61.3
cluster_id = test-cluster-9a4f
etc_hosts_entries =
192.168.61.2 api.test-cluster-9a4f.mydomain.com
console-openshift-console.apps.test-cluster-9a4f.mydomain.com
integrated-oauth-server-openshift-authentication.apps.test-cluster-9a4f.mydomain.com
oauth-openshift.apps.test-cluster-9a4f.mydomain.com
prometheus-k8s-openshift-monitoring.apps.test-cluster-9a4f.mydomain.com
grafana-openshift-monitoring.apps.test-cluster-9a4f.mydomain.com
bolsilludo.apps.test-cluster-9a4f.mydomain.com

install_status = COMPLETED
```

```
master_ips = [  
    "192.168.61.4",  
    "192.168.61.5",  
    "192.168.61.6",  
]  
oc_server_url = https://api.test-cluster-9a4f.mydomain.com:6443/  
storageclass_name = nfs-storage-provisioner  
web_console_url = https://console-openshift-console.apps.test-cluster-9a4f.mydomain.com  
worker_ips = []
```

These details can be retrieved anytime by running the following command from the root folder of the code:

```
$ terraform output
```

Post Install

After installation you need to do the following:

- ▶ Once the deployment is completed successfully, you can safely delete the bootstrap node. This step is optional but recommended so as to free up the resources used. The process is shown in “Clean up” on page 171.
- ▶ Configure the *dns* as explained in “Create API and Ingress DNS Records” on page 170

3.4.6 IBM Power Hardware Management Console as Ansible-managed client

Hardware Management Consoles can be Ansible clients, either using the HMC Collection that connects to the HMC using the https API, or by running the *ansible.builtin.shell* extension using `cmd/ssh`. This section discusses some of the setup considerations for setting up your HMC as an Ansible client.

Defining a user for Ansible

Its not a good practice to use *hscroot* to run your Ansible scripts. This section provides a quick guide to creating a user for your Ansible script and provide the correct roles and access capabilities.

Use the following steps to set up your user and roles.

1. Login to your HMC with a user with Admin rights and go to *Users and Security*.
2. Select *Users and Roles*.
3. Click on *Manage User Profiles and Access*, as shown in Figure 3-24 on page 174.
4. Fill in the required fields.
5. Select the *Managed Resource Roles*.
6. Choose the *Task Roles*.
7. You can choose to create new roles based on your specific requirements.

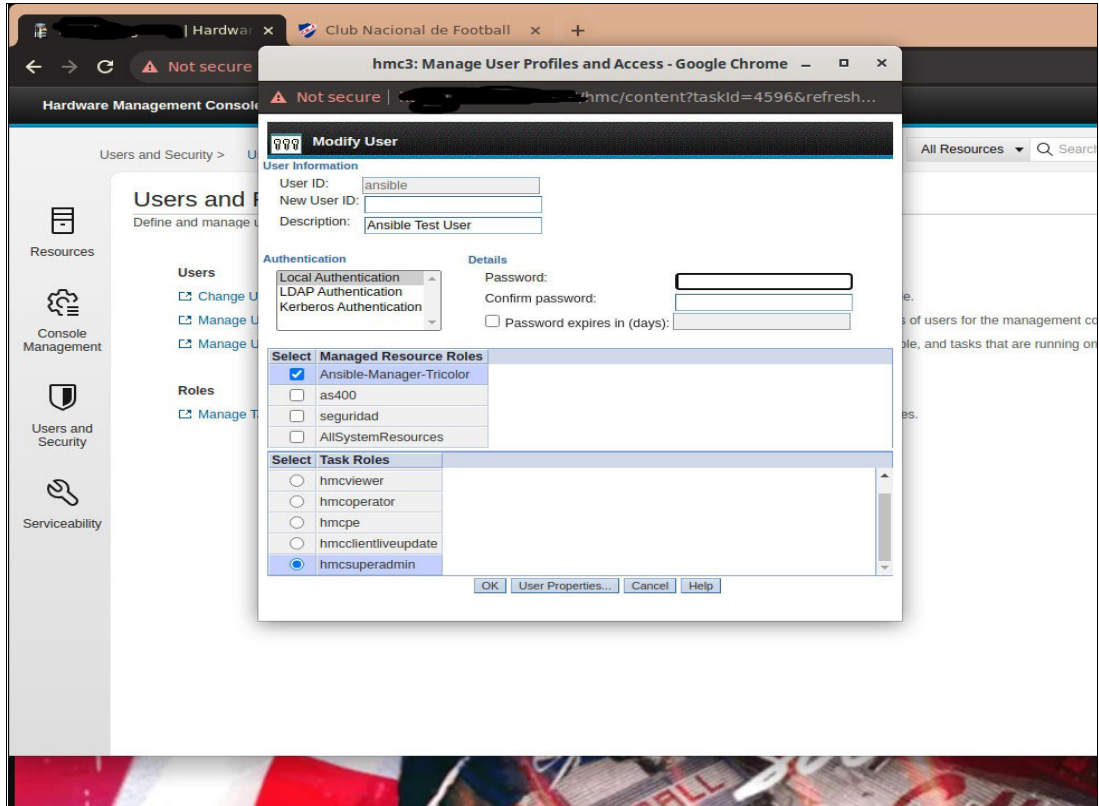


Figure 3-24 Create user Role in HMC

8. Next click on “User Properties” to enable remote access as shown in Figure 3-25.

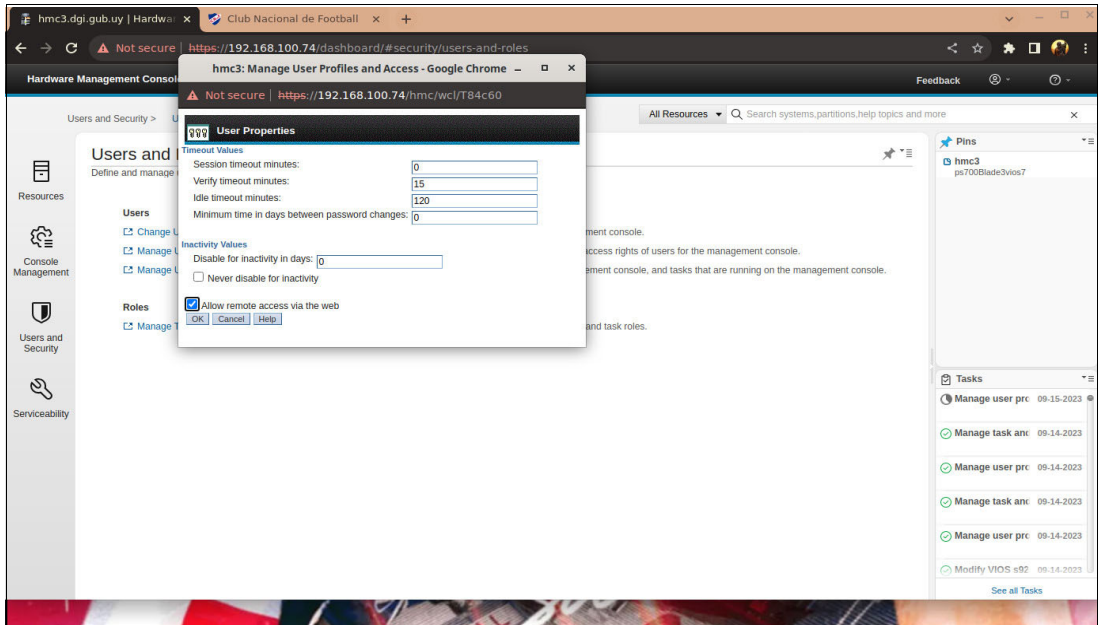


Figure 3-25 Enable remote access for User

9. Depending on your security requirements, you may need to modify the access privileges defining what resources (Frames and LPARs) that can be managed by your Ansible user. This information is defined in the user roles, for each role we can define access to a

reduced set of frames, and on those frames, a reduced set of machines. To illustrate this, in Figure 3-26 we created a role that is only allowed to manage a reduced set of frames and LPARs.

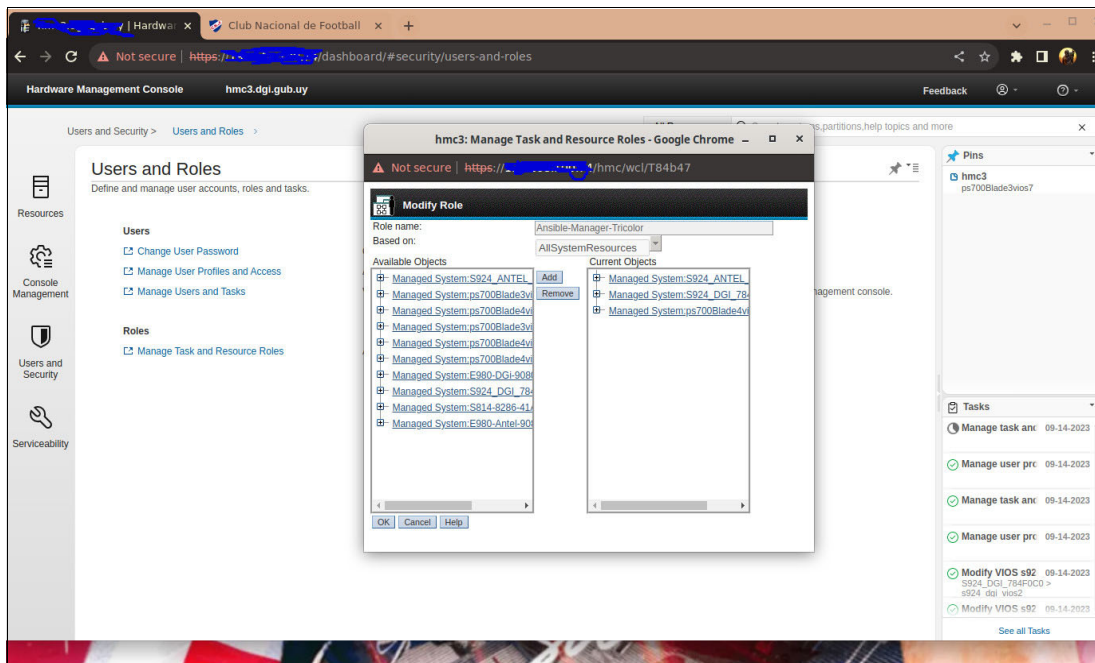


Figure 3-26 Create reduced ROLE

10.If you define a new role after creating the user, you can just modify the user and select the new Role as shown in Figure 3-27

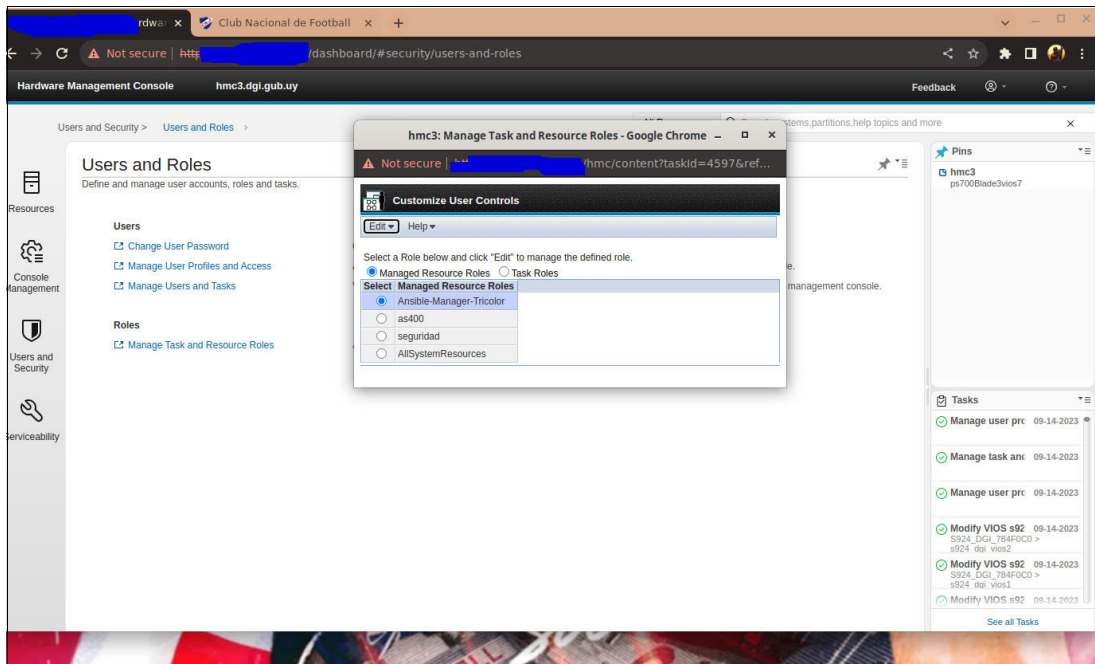


Figure 3-27 Select Role for User

At this point, we have setup a user with a set of permissions to run our Ansible scripts, through ssh or through the *hmc ibm power* collection.

Installing the collection

In order to be able to install the collection on your Ansible controller, please be sure to follow the installation instructions at

<https://ibm.github.io/ansible-power-hmc/installation.html>.

Along with the installation guide, we are providing some hints and tips that we collected during our testing.

1. Prior to running the install, make sure that you have setup python3.9 as defaults for python and pip3 as shown in Example 3-79.

Example 3-79 python install directory

```
[root@ansible-AAP-redbook ~]# alternatives --list | grep -i python
python          auto      /usr/libexec/no-python
python3         manual   /usr/bin/python3.9
root@ansible-AAP-redbook ~]# ls -lrt /usr/bin/python3
lrwxrwxrwx. 1 root root 25 Nov  3  2021 /usr/bin/python3 -> /etc/alternatives/python3
root@ansible-AAP-redbook ~]# ls -lrt /usr/bin/pip3
lrwxrwxrwx. 1 root root 22 Nov  3  2021 /usr/bin/pip3 -> /etc/alternatives/pip3
root@ansible-AAP-redbook ~]#
```

2. Run `ansible-galaxy`.

If you are behind a proxy you might need to run the `ansible` command with the `--ignore-certs` option. The collection installation progress will be output to the console. Note the location of the installation so that you can review other content included with the collection, such as the sample playbook. This is shown in Example 3-80.

Example 3-80 Install collection on Ansible controller node

```
[root@ansible-AAP-redbook ~]# ansible-galaxy collection install ibm.power_hmc
--ignore-certs
Starting galaxy collection install process
Process install dependency map
Starting collection install process
Downloading https://galaxy.ansible.com/download/ibm-power_hmc-1.8.0.tar.gz to
/root/.ansible/tmp/ansible-local-8811f_xjfbx/tmpgzq9rzj1/ibm-power_hmc-1.8.0-hi_qmsj1
Installing 'ibm.power_hmc:1.8.0' to
'/root/.ansible/collections/ansible_collections/ibm/power_hmc'
ibm.power_hmc:1.8.0 was installed successfully
root@ansible-AAP-redbook ~]#
```

If you want to install on a special directory you can run the install with the `-p` parameter to specify the install directory:

```
[root@ansible-AAP-redbook ~]# ansible-galaxy collection install
ibm.power_hmc --ignore-certs -p /home/ansible/collections
```

If everything finished OK you can check the install directory (instead of `/root`, the path will include the user directory tied to the user you used to run the install).

```
[root@ansible-AAP-redbook ~]# pwd
/root/.ansible/collections/ansible_collections/ibm/power_hmc/
```

Example 3-81 shows the content of the install directory in our example install.

Example 3-81 Structure of install directory

```
[root@ansible-AAP-redbook power_hmc]# ls -lrt
total 88
-rw-r--r--. 1 root root  947 Aug 22 15:14 MANIFEST.json
-rw-r--r--. 1 root root 35149 Aug 22 15:14 LICENSE
```

```
-rw-r--r--. 1 root root 16874 Aug 22 15:14 FILES.json
-rw-r--r--. 1 root root 2824 Aug 22 15:14 CONTRIBUTING.md
drwxr-xr-x. 5 root root 77 Aug 22 15:14 plugins
-rw-r--r--. 1 root root 263 Aug 22 15:14 MAINTAINERS.md
drwxr-xr-x. 2 root root 30 Aug 22 15:14 collections
-rw-r--r--. 1 root root 3227 Aug 22 15:14 CODE_OF_CONDUCT.md
drwxr-xr-x. 4 root root 32 Aug 22 15:14 tests
drwxr-xr-x. 4 root root 53 Aug 22 15:14 docs
-rw-r--r--. 1 root root 5 Aug 22 15:14 requirements.txt
drwxr-xr-x. 2 root root 25 Aug 22 15:14 meta
-rw-r--r--. 1 root root 1352 Aug 22 15:14 README.md
drwxr-xr-x. 3 root root 59 Aug 22 15:44 context
-rw-r--r--. 1 root root 144 Aug 22 16:51 execution-environment.yml
drwxr-xr-x. 2 root root 4096 Sep 15 11:55 playbooks
[root@ansible-AAP-redbook power_hmc]#
```

Connecting to the HMC

The collection connects to the HMC using the HMC API, however you can also use the Ansible native ssh connection to run HMC commands.

Example 3-82 shows an example of using the Ansible built in shell to run cli commands on the HMC.

Example 3-82 Using Ansible built in shell to manage HMCs

```
[root@ansible-AAP-redbook playbooks]# cat inventory
[hmc3]
hmc3
hmc4
hmc5

[hmc3:vars]
ansible_user=ansible
hmc_password=xxxxxx

[root@ansible-AAP-redbook playbooks]# cat hmc_lpar_limited.yml
---
- name: HMC List partition
  hosts: hmc3
  collections:
    - ibm.power_hmc
  connection: local
  vars:
    curr_hmc_auth:
      username: "{{ ansible_user }}"
      password: "{{ hmc_password }}"

  tasks:
    - name: Get Information about the Frames Installed on the HMC
      ansible.builtin.shell:
        cmd: "sshpass -p {{ hmc_password }} ssh {{ ansible_user }}@{{ inventory_hostname }}
1ssyscfg -r sys"
      register: config

    - name: Filter Output to get just the Frame Name
      set_fact:
        filtered_output: "{{ config.stdout | regex_findall('^name=(.*?),', '\\1') |
join(',') }}"
```



```

- name: Show Filtered Output
  debug:
    var: filtered_output

- name: Save Filtered Output to a File for Later use
  ansible.builtin.copy:
    content: "{{ filtered_output }}"
    dest: "/var/ansible/output/newout_{{ inventory_hostname }}.txt"

- name: List all the LPARs in the selected Frames
  ansible.builtin.shell:
    cmd: " sshpass -p {{ hmc_password }} ssh {{ ansible_user }}@{{ inventory_hostname
}} lssyscfg -r lpar -m {{ item }}"
    with_items: "{{ filtered_output.split(',') }}"
    register: lpar_info

- name: Save the Output, LPAR List for each Frame
  ansible.builtin.copy:
    content: "{{ lpar_info }}"
    dest: "/var/ansible/output/Lparout.txt"

- name: Show LPAR Information
  debug:
    var: item.stdout_lines
    with_items: "{{ lpar_info.results }}"

```

```
[root@ansible-AAP-redbook playbooks]#
```

In our playbook execution, the inventory has 3 HMCs but we only created the user with special permits in one of them, hmc3. When we ran the playbook, the others failed because the user does not exist. The output from the execution is shown in Example 3-83.

Example 3-83 Playbook execution

```

[root@ansible-AAP-redbook playbooks]# ansible-playbook -i inventory hmc_lpar_limited.yml

PLAY [HMC List partition] *****

TASK [Gathering Facts] *****
ok: [hmc3]
ok: [hmc5]
ok: [hmc4]

TASK [Get Information about the Frames Installed on the HMC] *****
changed: [hmc3]
fatal: [hmc5]: FAILED! => {"changed": true, "cmd": "sshpass -p XXXXXX ssh ansible@hmc5
lssyscfg -r sys", "delta": "0:00:01.867500", "end": "2023-09-18 15:42:07.700107", "msg":
"non-zero return code", "rc": 5, "start": "2023-09-18 15:42:05.832607", "stderr": "",
"stderr_lines": [], "stdout": "", "stdout_lines": []}
fatal: [hmc4]: FAILED! => {"changed": true, "cmd": "sshpass -p XXXXXX ssh ansible@hmc4
lssyscfg -r sys", "delta": "0:00:01.843471", "end": "2023-09-18 15:42:07.787177", "msg":
"non-zero return code", "rc": 5, "start": "2023-09-18 15:42:05.943706", "stderr": "",
"stderr_lines": [], "stdout": "", "stdout_lines": []}

TASK [Filter Output to get just the Frame Name] *****
ok: [hmc3]

TASK [Show filtered Output] *****
ok: [hmc3] => {
  "filtered_output": "S924_ANTEL_XXXX,S924_DGI_XXXX,ps700Blade4XXXX"
}

```



```

}

TASK [Save Filtered Output to a File for Later use] *****
ok: [hmc3]

TASK [List all the LPARs in the selected Frames] *****
changed: [hmc3] => (item=S924_ANTEL_XXXX)
changed: [hmc3] => (item=S924_DGI_784FOC0)
changed: [hmc3] => (item=ps700Blade4XXXX)

TASK [Save the Output, LPAR List for each Frame] *****
changed: [hmc3]

TASK [List with each LPAR Detailed Information] *****
ok: [hmc3] => {
  "lpar_info.stdout": "VARIABLE IS NOT DEFINED!"
}

TASK [Mostrar información de LPARs] *****
ok: [hmc3] => (item={'changed': True, 'stdout':
'name=ansibleRH8_1,lpar_id=27,lpar_env=aixlinux,state=Running,resource_config=1,os_version=
Unknown,logical_serial_num=XXXXX,default_profile=default_profile,curr_profile=default_profi
le,work_group_id=none,shared_proc_pool_util_auth=0,allow_perf_collection=0,power_ctrl_lpar_
ids=none,boot_mode=norm,lpar_keylock=norm,auto_start=0,redundant_err_path_reporting=0,rmc_s
tate=inactive,rmc_ipaddr=,time_ref=0,lpar_avail_priority=127,desired_lpar_proc_compat_mode=
default,curr_lpar_proc_compat_mode=POWER9_base,simplified_remote_restart_capable=0,sync_cur
r_profile=1,affinity_group_id=none,vtpm_enabled=0,migr_storage_vios_data_status=Data
Collected,migr_storage_vios_data_timestamp=Wed Sep 06 08:20:38 UTC
2023,powervm_mgmt_capable=0,pend_secure_boot=0,curr_secure_boot=0,keystore_kbytes=0,virtual
_serial_num=none\nname=ansibleRH8_2,lpar_id=28,lpar_env=aixlinux,state=Running,resource_con
fig=1,os_version=Unknown,logical_serial_num=XXXXX,default_profile=default_profile,curr_prof
ile=default_profile,work_group_id=none,sha
...
...
...
start=0,redundant_err_path_reporting=0,rmc_state=active,rmc_ipaddr=XXXXXX,lpar_avail_priori
ty=191,desired_lpar_proc_compat_mode=default,curr_lpar_proc_compat_mode=POWER7,sync_curr_pr
ofile=0,affinity_group_id=none,migr_storage_vios_data_status=unavailable,migr_storage_vios_
data_timestamp=unavailable"
  ]
}
PLAY RECAP *****
hmc3                : ok=9    changed=3    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
hmc4                : ok=1    changed=0    unreachable=0    failed=1    skipped=0
rescued=0    ignored=0
hmc5                : ok=1    changed=0    unreachable=0    failed=1    skipped=0
rescued=0    ignored=0

[root@ansible-AAP-redbook playbooks]#

```

This playbook was just an example of using the ssh connection to run cli commands. There are many other options using the HMC collection that provide an extended set of features.

Using the dynamic inventory plugin

The dynamic inventory plugin, `powervm_inventory`, helps to collect the inventory of all available virtual machines in the public or private cloud infrastructure managed by the HMC. This saves the administrator from maintaining a multitude of static inventory listings. This

dynamic plugin facilitates the consolidation of the partitions into various groups and it can be further fine-tuned according to its property. That means you can now dynamically group and manage these partitions according to data center policy.

An example use case can be to apply specific patches only to a targeted version of AIX partition by using the respective group created by this dynamic inventory management plugin. The playbook shown in Example 3-84 is an example of dynamically creating a group definition from the LPARs being managed. The playbook dynamically creates a group of all the running AIX partitions with OS version 7.2,. This can be used as an inventory input to perform patching or for OS upgrades.

Example 3-84 Dynamic inventory example

```
plugin: ibm.power_hmc.powervm_inventory
hmc_hosts:
  - hmc: <hmc_host_name_or_IP>
    user: <hmc_username>
    password: <hmc_password>
filters:
  PartitionState: 'running'
groups:
  AIX_72: "'7.2' in OperatingSystemVersion"
```

Patch management of HMC

The patch management module, *hmc_update_upgrade*, is essentially designed to support all the update and upgrade requirements of HMC. The module supports centralized patch management. The upgrade files, service packs and security fixes in can be stored in SFTP, FTP, or NFS servers or the HMC images can be kept on the Ansible control node and can be configured as the image source for patch management. Example 3-85 shows a playbook for patching or upgrading the HMC.

Example 3-85

```
- name: Upgrade the HMC from 910 to 950
  hosts: hmcs
  collections:
    - ibm.power_hmc
  connection: local
  vars:
    curr_hmc_auth:
      username: <hmc_username>
      password: <hmc_password>

  tasks:
    - name: Update the 910 HMC with 9.1.910.6 PTF
      hmc_update_upgrade:
        hmc_host: '{{ inventory_hostname }}'
        hmc_auth: '{{ curr_hmc_auth }}'
        build_config:
          location_type: ftp
          hostname: <FTP_Server_IP>
          userid: <FTP_Server_uname>
          passwd: <FTP_Server_pwd>
          build_file: HMC9.1.910.6/2010170040/x86_64/MH01857-9.1.910.6-2010170040-x86_64.iso
          state: updated

    - name: Upgrade the 910 HMC with 950 image
      hmc_update_upgrade:
        hmc_host: '{{ inventory_hostname }}'
```

```
hmc_auth: '{{ curr_hmc_auth }}'  
build_config:  
location_type: nfs  
hostname: <NFS_Hostname/IP>  
mount_location: <upgrade_img_mount_loc>  
build_file: /HMC9.2.950.0/2010230054/x86_64/network_install  
state: upgraded
```

In this snippet of a playbook there are two tasks defined utilizing the *hmc_update_upgrade* module. The first task takes uses a PTF image stored in a FTP server to upgrade the HMC running 9.1.910 level and the second task upgrades the HMC to 9.2.950 level using an image stored in NFS server. Note that to update an HMC the task state should be defined as *updated* and to upgrade an HMC the task state should be defined as *upgraded*. Both states support NFS, FTP, SFTP and Disk (keeping the image in controller node) HMC image repositories.

Additional examples can be found [on this blog](#). More examples and samples are included in the collection and can be found at <https://ibm.github.io/ansible-power-hmc/index.html>



Automated Application Deployment on Power Servers

One of the areas that can benefit the most from automation with Ansible is the area of application deployment. This is true, not only for the applications that your application development team are building specifically for your business operations, but also is true for implementing common application environments like Oracle and SAP.

For application development, automation can help you build a modern continuous integration and continuous development (CI/CD) pipeline which provides a management structure around your development cycle and allows you to be more agile by integrating application changes quickly and safely.

For application and database environments like Oracle and SAP, there are often multiple instances of those environments that need to be deployed and maintained, often doing repetitive tasks like building virtual machines and applying updates to your existing environments. Automation using Ansible can help your IT specialists perform those repetitive functions in an efficient, managed and repeatable way – freeing up time to do other tasks that provide better value to your business.

The following topics are discussed in this chapter...

- ▶ Deploying and managing applications using Ansible on Power servers
- ▶ Automated Application Deployment on Power Servers
- ▶ Deploying a simple Node.js application
- ▶ Orchestrating multi-tier application deployments
- ▶ Continuous integration and deployment pipelines with Ansible
- ▶ Oracle DB automation on Power Systems
- ▶ SAP automation

4.1 Deploying and managing applications using Ansible on Power servers

Ansible can be used to automate many of the facets of your application environment, both on IBM Power and on other platforms. Ansible can assist you in automating your application development environments, allowing you to create pipelines that assist in managing the development of code, the testing of the code and the integration of the new code into your application environment in a safe and efficient way.

Ansible can also automate the installation and management of many middleware and application products such as Oracle Database, SAP NetWeaver and SAP HANA workloads, IBM DB2 databases, and Red Hat OpenShift deployments.

4.2 Automated Application Deployment on Power Servers

The IBM Power Systems platform is rapidly evolving, with major advances being made across IBM AIX, IBM i, and Linux on Power. Hybrid multicloud demands consistency and agility from all of these platforms. Today's IT administrators, developers, and QA engineers want to streamline anything they can to save time and increase reliability.

4.2.1 Ansible Content for IBM Power Systems

IBM Power Systems is a family of enterprise servers that helps transform your organization by delivering industry leading resilience, scalability and accelerated performance for the most sensitive, mission critical workloads and next-generation AI and edge solutions. The Power platform also leverages open source technologies that enable you to run these workloads in a hybrid cloud environment with consistent tools, processes and skills.

4.2.2 IBM AIX, IBM i, and Linux on Power Collections for Ansible

The IBM Power Systems AIX collection provides modules that can be used to manage configurations and deployments of Power AIX systems. Similar functions are provided by the Power IBM i collection and the Linux on Power collection. The content in these collections helps to manage applications and workloads running on IBM Power platforms as part of an enterprise automation strategy within the Ansible ecosystem. These collections are available from Ansible Galaxy with community support or through Red Hat Ansible Automation Platform with full support from Red Hat and IBM.

To show the power and utility of using Ansible to manage your applications in an IBM Power environment, we provide three use cases:

- 4.3, “Deploying a simple Node.js application” on page 185 show you how you can automate application deployment in an AIX environment
- 4.4, “Orchestrating multi-tier application deployments” on page 186 points you to a tutorial on orchestrating a two tier application with an application tier and a database tier.
- 4.5, “Continuous integration and deployment pipelines with Ansible” on page 186 shows how to you can use Ansible to automate a CI/CD environment for an application in an IBM i environment.

4.3 Deploying a simple Node.js application

This section shows how to install a sample Node.js application on AIX and then start that application on the host utilizing Ansible. The Ansible playbook to accomplish this is shown in Example 4-1.

Example 4-1 Install Node.js application and start the application

```

---
- hosts: all
  gather_facts: false
  collections:
    - ibm.power_ibmi

  vars:
    checkout_dir: 'tmp_nodejs'

  tasks:
    - name: Install Node js
      command: /Q0pensys/pkgs/bin/yum install nodejs10 -y
      ignore_errors: true

    - name: verify git has been installed
      stat:
        path: /Q0pensys/pkgs/bin/git
      register: git_stat

    - name: Install git if it is not there
      command: /Q0pensys/pkgs/bin/yum install git -y
      when: not git_stat.stat.exists

    - name: upgrade yum in case EC_POINT_copy error
      command: /Q0pensys/pkgs/bin/yum upgrade -y

    - name: create symlink for git command to use git module
      command: ln -fs /Q0pensys/pkgs/bin/git /usr/bin/git
      ignore_errors: true

    - name: set http.sslVerify for git
      command: 'git config --global http.sslVerify false'
      ignore_errors: true

    - name: clone repo
      git:
        repo: 'https://github.com/IBM/ibmi-oss-examples.git'
        dest: '{{ checkout_dir }}'

    - name: npm i
      shell: "(/Q0pensys/pkgs/lib/nodejs10/bin/npm i --scripts-prepend-node-path)"
      args:
        warn: false
        chdir: '{{ checkout_dir }}/nodejs/mynodeapp'
        executable: /usr/bin/sh

    - name: Start the demo application
      shell: "(nohup /Q0pensys/pkgs/lib/nodejs10/bin/node index.js >/dev/null 2>&1 &)"
      args:
        warn: false
        chdir: '{{ checkout_dir }}/nodejs/mynodeapp'
        executable: /usr/bin/sh
        async: 10

```

The playbook first validates that the required infrastructure components (git and yum) to retrieve the application and install it. It then retrieves the application from the git repository and has Ansible start the application on the server.

4.4 Orchestrating multi-tier application deployments

In the era of microservices, containers, Kubernetes, and DevOps, deployment of complex multi-tier systems in the public cloud is achieved with continuous integration and delivery (CI/CD) pipelines. However, things become complicated when attempting to deliver applications onto private cloud or on-premise systems, where CI/CD isn't owned by development or doesn't even exist at all.

4.4.1 Orchestration in the world of Kubernetes

Deploying complex application systems is not a new problem. Over the past few decades, the need for automated configuration and management – or orchestration – of software has been identified many times. In the operating systems space, configuration management tools like Chef, Puppet, Salt, and finally, Ansible, orchestrate configuration of OS-native applications.

From an application orchestration standpoint, many prominent players have emerged in the Kubernetes space such as: viz. Helm, Operator Framework, Kustomize, Automation, Broker and Ansible Kubernetes module.

IBM has developed a step by step tutorial showing the deployment and orchestration of a sample *WordPress* application along with its dependent MySQL database using Ansible. The tutorial can be found at the following link:

<https://developer.ibm.com/tutorials/orchestrate-multi-tier-application-deployments-on-kubernetes-using-ansible-pt-3/>

4.5 Continuous integration and deployment pipelines with Ansible

Ansible can help you automate the continuous integration and continuous deployment (CI/CD) cycle for your application. The CI part of the CI/CD involves automatic versioning control for your application code. As changes are made in your code, Ansible can automatically save those changes in a repository so that you know what changes were made and when. To enable the CD component, you must also have automation in your CI components that provides testing and validation of new versions of your code. You likely will initially apply the new code into a test or quality assurance (QA) environment prior to moving the code to production. Ansible can assist in all of those processes.

4.5.1 CI/CD using Ansible for IBM i

Continuous Integration (CI) and Continuous deployment (CD) stand as pivotal pillars within the software development landscape. Ansible, an automation tool, harmonizes with these practices, aligning software development with a unified approach. CI revolves around frequent code integration, while CD automates deployment, ensuring swift and reliable delivery. Ansible effectively supports these practices, orchestrating tasks and interactions through various stages.

For IBM i users, Ansible emerges as a versatile solution for crafting CI/CD pipelines. It interfaces with Version Control Solutions such as GitHub, facilitating tasks ranging from code validation to final deployment. Ansible's utility spans beyond Continuous Testing, merging into the broader domain of development processes. This adaptability benefits both native applications and open-source software deployment on the IBM i platform.

The foundation of a CI/CD pipeline through Ansible takes shape as a series of interconnected stages. Starting with the Development Environment, Ansible playbooks automate tasks such as provisioning IBM i virtual machines, code retrieval, build and deployment, and branching for new development cycles. This initial phase lays the groundwork for efficient development practices.

In the second stage, Developers play a hands-on role in code modifications and unit testing. In this phase, developers make code changes, conduct unit tests on the development IBM i system, commit code alterations to specific branches, and create pull requests. This phase demonstrates the collaborative essence of the development process.

As the CI/CD pipeline progresses, Ansible again takes center stage during the Testing phase. Ansible playbooks orchestrate the setup of new IBM i virtual machines for testing, the installation of dependencies, cloning development branches, and the execution of comprehensive tests. Furthermore, Ansible assists in pull request approval and automated merge processes, ensuring code integration. Upon test completion, Ansible manages environment cleanup for ongoing efficiency.

The final phase, Deployment, encapsulates the core objective of delivering refined software to production environments. Ansible playbooks manage cloning the latest code from master branches to build systems, directing the build process, facilitating deployment to production systems, and maintaining environment hygiene through cleanup procedures. This phase marks the realization of the CI/CD pipeline, where developed and tested code becomes accessible to end-users.

Figure 4-1 shows a diagram describing the four stages described above for CI/CD using Ansible:

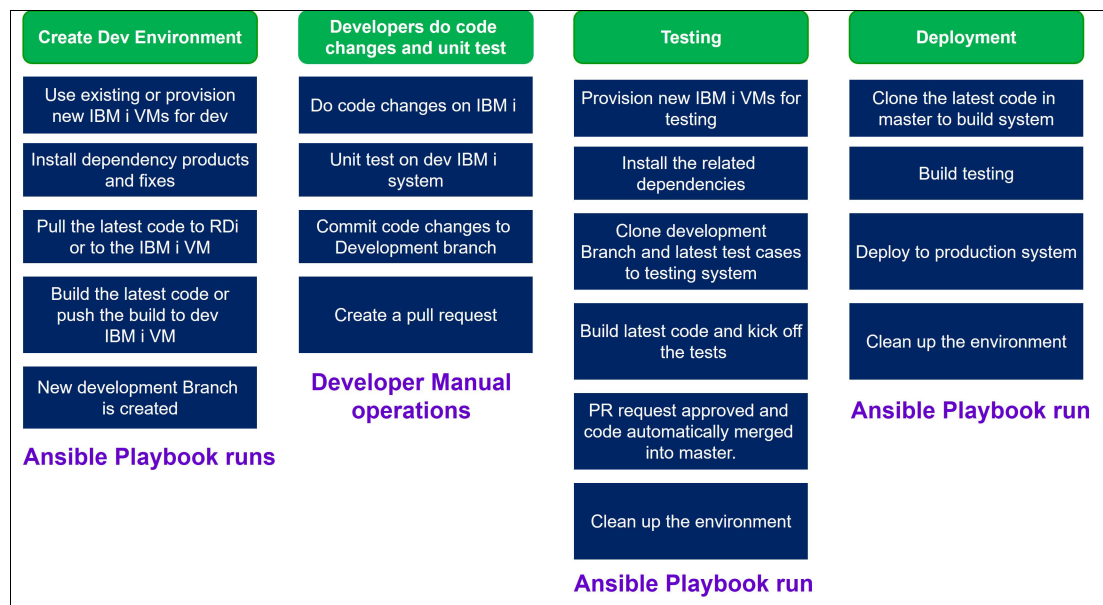


Figure 4-1 Integrated CI/CD Pipeline orchestrated by Ansible

To display some sample code showing how to automate your CI/CD processes with Ansible, go to “Full cycle of the CI/CD process on IBM i with Ansible” on page 344.

Note: Ansible's role within the CI/CD pipeline significantly streamlines IBM i application development. Its automation capabilities extend across development environment creation, manual developer tasks, testing orchestration, and deployment management. By aligning with contemporary CI/CD practices, Ansible contributes to advancing IBM i software development with heightened speed, dependability, and agility.

4.6 Oracle DB automation on Power Systems

There are three collections available for managing your Oracle Database with Ansible on AIX.

The single-node collection supports both JFS-based installation and ASM-based installation while the Oracle RAC collection supports a multi-layered installation process. Such layers include infrastructure provisioning with PowerVC, and grid setup configuration and installation, as well as setup and installation of the database binaries.

If you are not using PowerVC to help you manage your environment, you can still use the collection to install and setup the grid and database installation. Doing so requires manual setup of the nodes as per Oracle RAC requirements.

The Oracle DBA collection provides a set of tools to enable database administrators to automate a wide range of their administrative activities, such as patching the database servers, creating new database instances, managing users, jobs, and tablespaces amongst a wide range of operations.

For specifics on the use of each of these collections see the following:

- ▶ 4.6.2, “Automating deployment of a single node Oracle database with Ansible” on page 189.
- ▶ 4.6.3, “Automating deployment of Oracle RAC with Ansible” on page 193.
- ▶ 4.6.4, “Automating Oracle DBA operations” on page 204

4.6.1 Why businesses opt for AIX to host their databases

We previously described in section 1.4.4, “Key benefits of IBM Power compared to x86 servers” on page 28 how IBM Power holds an upper hand in hosting applications and middleware in general. Furthermore, IBM Power offers better economics, and is highly flexible – allowing clients to build out a platform on which they can consolidate any number of applications and environments including databases.

Working together to provide services for their common clients, IBM and Oracle have established an alliance that shows a shared commitment to the success of those clients. As a result of this alliance, IBM and Oracle have over 80,000 joint clients that are provided with enhanced hardware and software support. This is enabled via an in-depth certification of Oracle database on AIX as a collective effort. The alliance provides an award-winning services practice, with a diamond partnership, as a result of extensive technology collaboration and cooperative client support.

For further details on why IBM Power Systems and AIX platform better suit hosting Oracle database, you can refer to *Oracle on IBM Power Systems*, SG24-8485.

4.6.2 Automating deployment of a single node Oracle database with Ansible

You can use Ansible to automate the deployment of a single-node Oracle database instance. It can be a filesystem-based installation, or ASM-based installation. The hosting AIX LPAR and database storage volumes may be created beforehand as setup for this installation or can be deployed by Ansible as shown in Chapter 5, “Infrastructure as Code Using Ansible” on page 227.

Figure 4-2 shows a high level overview of Oracle DB single-node deployment.

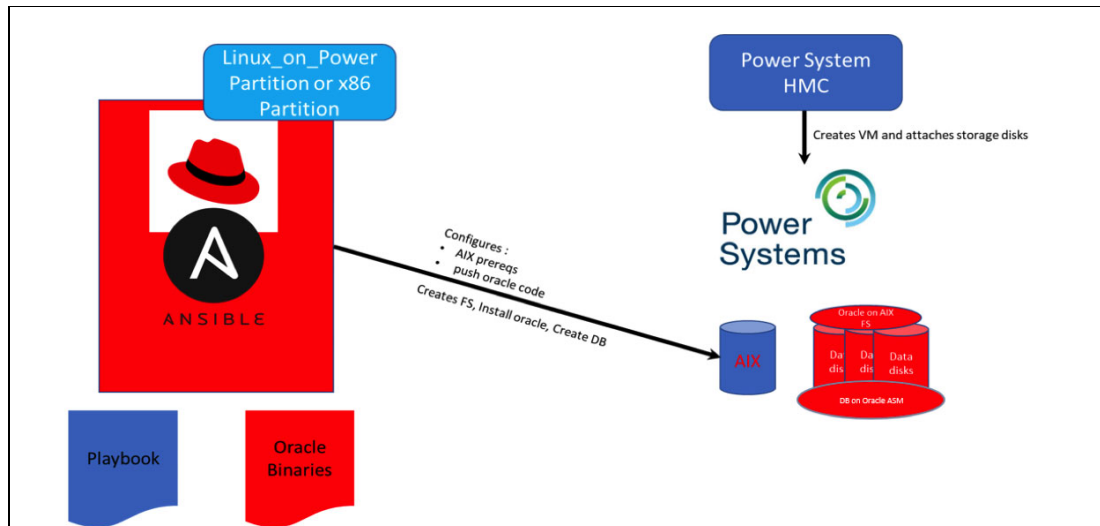


Figure 4-2 Single-node Oracle DB deployment topology¹

The Ansible `power_aix_oracle` galaxy collection contains artifacts (roles, vars, config files and playbooks) that are usable for the single-node Oracle database installation. Those artifacts have a number of prerequisites that need to be met prior to using the collection.

Single-node Oracle database installation prerequisites

The following prerequisites need to be met prior to running an Ansible playbook to automate the installation of a single-node Oracle database server.

1. A new LPAR running AIX 7.2 or 7.3.
 - a. The LPAR may be created using Ansible Infrastructure as Code methodology as described in Chapter 5, “Infrastructure as Code Using Ansible” on page 227.
 - b. The rootvg disk should be at least 30GB in size out of which `/tmp` filesystem should be 8GB as it will be used for Ansible remote location. Also, the paging space device must be adjusted to use `rootvg`.
 - c. Two disks of the following sizes:
 - i. 40GB for filesystem-based installation or 75GB for ASM-based installation.
This disk is intended to host the database binaries for installation types as well as the grid binaries for the ASM-based installation.
 - ii. At least 20GB for the creation of a test Oracle database instance post successful installation.

¹ https://github.com/IBM/ansible-power-aix-oracle/blob/main/docs/README_ORA_SI_Play.pdf

- d. Disks in c on page 189 above should be clean from any old header data.

To check the header info use the following command:

```
lquerypv -h /dev/hdiskX
```

If you need to clear the disk's pvid use this command:

```
chdev -l hdiskX -a pv=clear
```

To clear the header data, use this command:

```
dd if=/dev/zero of=/dev/hdiskX bs=1024k count=100
```

- The Oracle database version currently supported on AIX is 19c. All power_aix_oracle collection artifacts assume this version for the installation process. It can be downloaded from [Oracle edelivery website](#) or [Oracle Technology Network \(OTN\)](#).
- The power_aix_oracle collection uses the [power_aix collection](#) for some of the configuration requirements, hence it needs to be installed as well on the Ansible server.

Note: If the LPAR has not been bootstrapped for Ansible before, it may be a good idea to do so now.

To bootstrap an LPAR for Ansible follow these steps:

- Ensure that ibm.power_aix collection is installed.
- Copy `demo_bootstrap.yml` from the playbooks directory of the collection to your workstation.
- Execute that playbook against your LPAR to boot strap it. `ansible-playbook demo_bootstrap.yml -l mylpar`.

Install power_aix_oracle collection

Use the command provided in Example 4-2 to download and install the collection.

Example 4-2 Installing the ibm.power_aix_oracle collection from Ansible Galaxy website

```
ansible-galaxy collection install ibm.power_aix_oracle
Process install dependency map
Starting collection install process
Installing 'ibm.power_aix_oracle:1.1.1' to
'/root/.ansible/collections/ansible_collections/ibm/power_aix_oracle'
```

Once the collection is installed successfully, all components such as roles, variables, and playbooks are saved in the directory specified in the last output line of Example 4-2. You may copy that directory to a working directory to update it with your variables and other parameters to avoid changing the installed source.

Figure 4-3 show the commands to copy the collection and then shows the contents of the collection.

```
root@ansible ~# pwd
/root
root@ansible ~# mkdir workspace
root@ansible ~# cp -r /root/.ansible/collections/ansible_collections/ibm/power_aix_oracle/ workspace/
root@ansible ~# cd workspace/power_aix_oracle/; ls
CODE_OF_CONDUCT.md  LICENSE          README.md        collections      inventory  roles
CONTRIBUTING.md    MAINTAINERS.md  _config.yml     demo_play_aix_oracle.yml  meta      tests
FILES.json          MANIFEST.json   ansible.cfg     docs             pics      vars
root@ansible power_aix_oracle#
```

Figure 4-3 Copy the collection's directory into a working directory and show its contents

The heavy lifting of the collection is done by the roles. There are three key roles which are stored in the roles directory as seen in Figure 4-3 on page 190.

The power_aix_oracle collection roles

There are three roles used in the power_aix_oracle collection:

► The **preconfig** role

The preconfig role performs AIX configuration tasks that are needed for oracle installation. It checks whether requirements are met and if they are not, it attempts to remedy them. Such checks include:

- Main operating system (rootvg) filesystems meet the minimum requirements.
- AIX filesets required by Oracle database such as bos.adt.*, bos.perf.*, rsct.* and XIC.* among others.
- RPM packages for tools used during the installation.
- Network configuration such as **hostname**, **real_hostname**, corresponding IP address and DNS server definition.
- Validate the disks to be used for the Oracle database installation, depending on whether it is filesystem-based or ASM-based installation.
- AIX tuning such as maxuproc, iocp device and paging space.
- Reboot to enact modified configurations.

► The **oracle_install** role

This role performs oracle binary installation. It sets the necessary components based on whether it is a filesystem or ASM installation, copy the binaries over to the AIX LPAR and installs them. It follows the sequence below:

- Set the execution variables including *grid_asm_flag* to *true* for ASM-based installation or to *false* for filesystem-based installation.
- Create the OS-level group and user necessary for the Oracle database installation (e.g. oper group and oinstall user).
- For filesystem-based installation:
 - Create volume group on the database disks.
 - Create and mount the filesystems at the correct sizes in the volume group created above.
- For ASM-based installation follow these steps:
 - Create Oracle grid home directory.
 - Set the ownership and access mode correctly for the ASM disks.
 - Setup grid source files and rootpre script.
 - Generate the grid response file and use it for the grid installation.
 - Run the **orainstallroot** and the **root** scripts.
 - Run the grid **ConfigTools** script.
- Perform the Oracle database installation using these steps:
 - Setup the *oinstall* user profile by pointing to the correct *tmp* directory and defining the oracle SID and oracle home directory.
 - Copy the Oracle installation binaries over from the repository to the LPAR,
 - Generate the Oracle installation response file and use it to install the binaries

► The **oracle_createdb** role

This role is used to create database instance(s) on the previously installed database server. It identifies whether the database server uses ASM storage or filesystem storage and runs the corresponding routine to create the database instance accordingly. It also uses variables such as target instance SID, password and character set defined in the variables for the creation.

To create an Oracle database instance on a server that uses ASM storage:

- Ensure no database instance exists with the same target SID.
- Generate the database instance creation template/script.
- Use the template/script to create the database instance.

To create an Oracle database instance on a server that uses filesystem storage:

- Ensure no database instance exists with the same target SID.
- Create a volume group for the database instance using an available volume set for the database.
- Create and mount a filesystem for the database instance on the volume group created in the previous step and ensure its ownership and access mode.
- Generate the database instance creation template/script.
- Use the template/script to create the database instance.

Steps to install Oracle DB 19c on AIX and create a database instance

1. Ensure all prerequisites documented in “Single-node Oracle database installation prerequisites” on page 189 are met including setting up the AIX LPAR, installing the `ibm.power_aix` collection and downloading the Oracle database installation software.
2. Setup the AIX LPAR as an Ansible client by adding it to the inventory file and exchange the Ansible server’s ssh key with it.
3. Ensure `ibm.power_aix_oracle` collection is installed as shown in Example 4-2 on page 190.
4. Optionally copy the `ibm.power_aix_oracle` directory to a temp working directory as shown in Figure 4-3 on page 190.
5. Copy `netns.conf` and `resolv.conf` from `/etc` directory of the AIX LPAR into the `roles/preconfig/files/` directory located inside the `workspace/power_aix_oracle/roles/preconfig/files` directory, where “workspace” is the temporary working directory you copied the collection directory to in step 4.
6. Modify the Oracle binaries location path variable “`oracle_dbaix19c`” in file “`workspace/power_aix_oracle/vars/oracle_params.yml`” and set the `grid_asm_flag` flag value to `true` for ASM-based installation or `false` for filesystem-based installation (the default option is filesystem-based). Go through all other parameters in that `oracle_params.yml` variables file and modify it as appropriate for your environment.
7. Run the playbook `demo_play_aix_oracle.yml` which essentially includes the variables file and then executes the 3 roles sequentially.

Example 4-3 on page 193 shows a playbook (**ansible-playbook demo_play_aix_oracle.yml**) that installs the Oracle database and creates a database instance.

Example 4-3 A playbook to install Oracle DB and creates an instances based on the vars file

```

- hosts: all
  gather_facts: yes
  vars_files: vars/oracle_params.yml
  roles:
    - role: preconfig
      tags: preconfig
    - role: oracle_install
      tags: oracle_install
    - role: oracle_createdb
      tags: oracle_createdb

```

To check on future updates of the collection, you may refer to its [documentation site](#).

4.6.3 Automating deployment of Oracle RAC with Ansible

Oracle RAC (Real Application Cluster) allows customers to run a single Oracle Database across multiple servers in order to maximize availability and enable horizontal scalability, while accessing shared storage.² IBM AIX is one of the most commonly used platform for hosting Oracle RAC.

Figure 4-4 shows a high level overview of Oracle RAC installation on existing infrastructure.

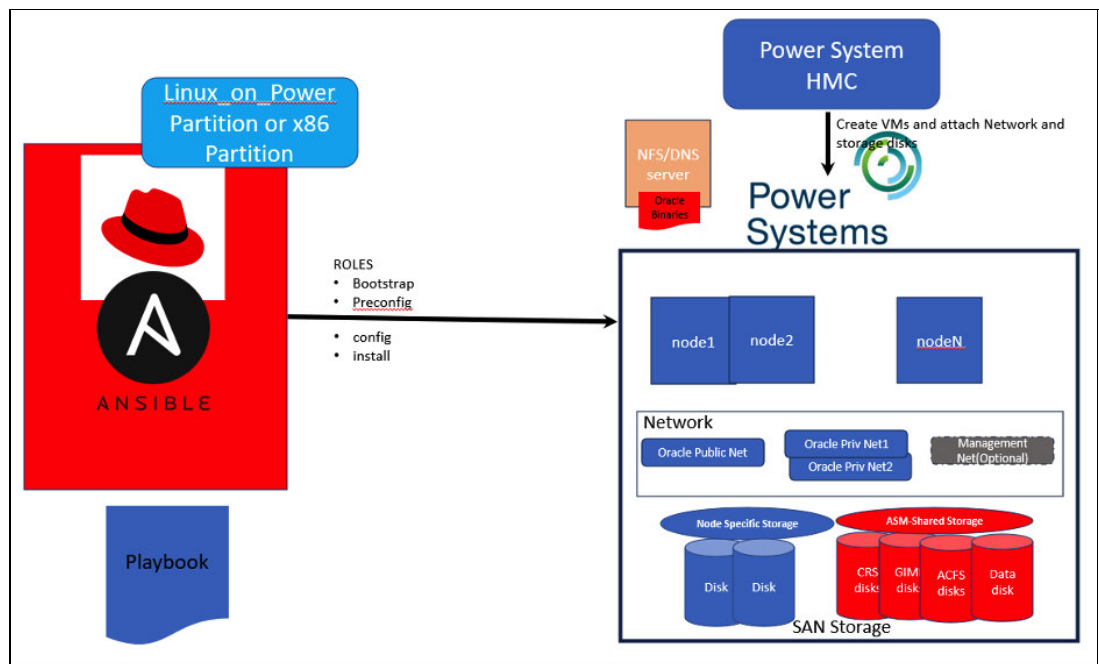


Figure 4-4 Oracle RAC deployment topology on existing infrastructure³

There is a video demonstration of using Ansible to install Oracle RAC created by one of the authors of this book. It can be found at <https://lnkd.in/d7Rjuupg>.

Figure 4-5 on page 194 shows the same overview for both infrastructure provisioning and Oracle RAC software installation automation.

² <https://www.oracle.com/qa/database/real-application-clusters/>

³ https://github.com/IBM/power-aix-oracle-rac-asm/blob/main/docs/README_ORACLE_RAC_PLAYBOOK_V1.3.pdf

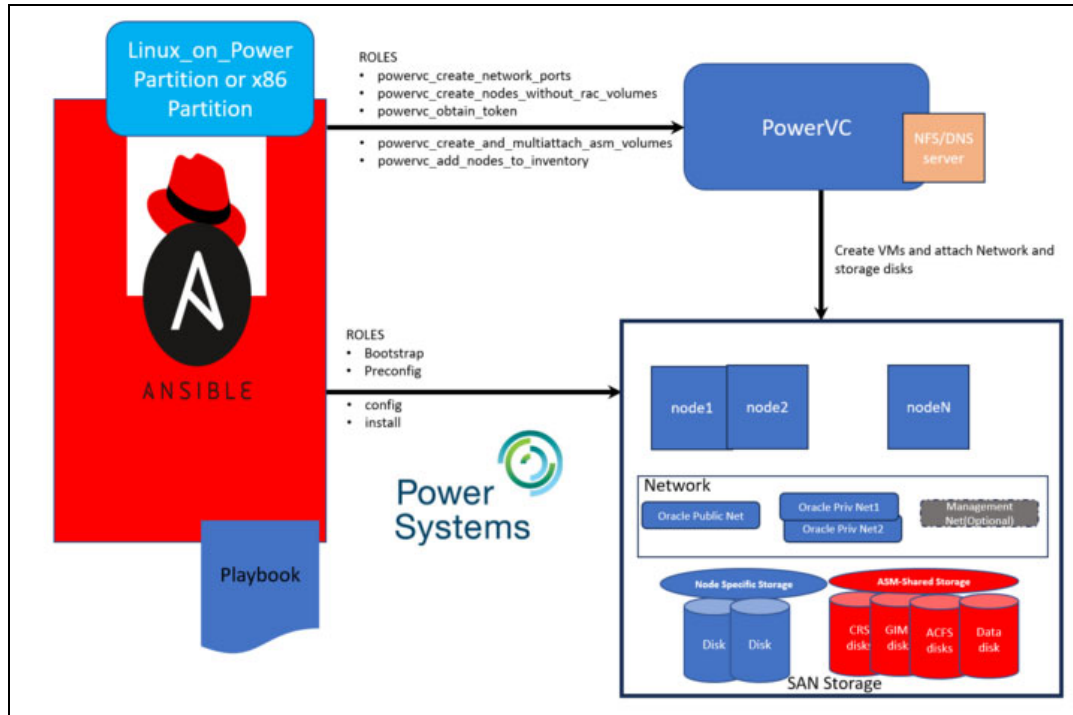


Figure 4-5 Oracle RAC deployment topology with IBM PowerVC automating the infrastructure layer⁴

Oracle RAC installation, on AIX and elsewhere, offers a wide range of complexities both at the infrastructure layer setup and in the software installation requirements. At the infrastructure layer the complexities encompass setting up the AIX nodes on hosts that meet the RAC requirements. These requirements include setting kernel tunable parameters, setting network attributes, setting shared disks attributes, and setting up ssh password-less access among others. It is a tedious, repetitive and error-prone process when done manually. Similarly, the manual process of the grid and database software installation is interactive and requires user's attentive physical presence for hours.

Luckily, Ansible automation is here for the rescue. The Ansible Oracle RAC collection (`ibm.power_aix_oracle_rac_asm`) available in [Ansible galaxy](#) and [GitHub](#), simplifies the installation of Oracle RAC 19c on the AIX operating system running on IBM Power servers by automating both the infrastructure setup operation and the software installation and configuration operation. It contains playbooks and a number of supporting roles and other artifacts that automate both layers.

The infrastructure layer automation via the collection requires IBM PowerVC. If your environment is not equipped with PowerVC, you can prepare the infrastructure manually and then use the collection for the Oracle RAC software installation. Otherwise, you can use the collection to automate both layers of the process.

Setup Ansible Server for Oracle RAC installation automation

This section covers Ansible server requirements for installing the Oracle RAC collection along with additional Ansible server configurations required for that collection.

⁴https://github.com/IBM/power-aix-oracle-rac-asm/blob/main/docs/README_ORACLE_RAC_PLAYBOOK_V1.3.pdf

Ansible server requirements for Oracle RAC collection

- ▶ The Ansible server must be version 2.9 or higher.
- ▶ The *python3-netaddr*, *perl*, *expect*, and *wget* RPMs must be installed.
- ▶ The *ibm.power_aix* collection (see 1.5.2, “Ansible for AIX” on page 38) must be installed because the Oracle RAC collection uses some of its modules in the setup process.
- ▶ The *ansible.utils* collection is required for the Oracle RAC collection to work. If it is not installed then use **ansible-galaxy collection install ansible.utils** command to install it.
- ▶ The [OpenStack sdk](#) must be installed when automating the provisioning of the infrastructure layer.

Troubleshooting note: There was a scenario where password-less authentication between the nodes failed with security-like errors which pointed to a missing *python3-selinux* RPM on the Ansible server. Disabling SELinux in the Ansible server resolved that issue. This will provide a workaround until a permanent resolution is reached.

Install power_aix_oracle_rac_asm collection

Use the example provided in Example 4-4 to download and install the collection. If your Ansible server does not have access to the Internet, then download the collection’s tarball to your workstation, transfer it to the Ansible server and then run the command with the same syntax but point to the tarball file instead.

Example 4-4 Install power_aix_oracle_rac_asm collection

```
ansible-galaxy collection install ibm.power_aix_oracle_rac_asm
Process install dependency map
Starting collection install process
Installing 'ibm.power_aix_oracle_rac_asm:1.2.1' to
'/root/.ansible/collections/ansible_collections/ibm/power_aix_oracle_rac_asm'
```

Once the collection is installed successfully, all of its contents will be stored in the directory shown in the last line of Example 4-4’s output above. Copy that directory to your workspace and work with it. This way, the original collection’s directory will remain as an unchanged reference.

Figure 4-6 shows contents of the collection’s directory post copying to the workspace

```
root@ansible workspace# pwd
/root/workspace
root@ansible workspace# cp -r ~/.ansible/collections/ansible_collections/ibm/power_aix_oracle_rac_asm/ .
root@ansible workspace# cd power_aix_oracle_rac_asm; ls
CODE_OF_CONDUCT.md  _config.yml          inventory
CONTRIBUTING.md    ansible-navigator.yml meta
MAINTAINERS.md      ansible.cfg           powervc_build_AIX_RAC_nodes.yml
MANIFEST.json       collections           roles
README.md           install_and_configure_Oracle_RAC.yml vars
root@ansible power_aix_oracle_rac_asm#
```

Figure 4-6 Copy the Oracle RAC installation collection directory to the workspace and show its contents

The files *powervc_build_AIX_RAC_nodes.yml* and *install_and_configure_Oracle_RAC.yml* are the playbooks used for automating the infrastructure layer and the Oracle RAC installation respectively. The *roles* directory hosts the roles supporting those playbooks and the *vars* directory contains the files where the variables used by these playbooks and roles are set.

Automate infrastructure layer provisioning with PowerVC

The infrastructure layer automation involves the activities below which are automated via the *powervc_build_AIX_RAC_nodes.yml* playbook and its supportive Ansible roles provided in the collection.

- ▶ An operational PowerVC server version 2.1.1 or newer.
- ▶ Setting up a PowerVC image configured with Oracle RAC specifications.
- ▶ Setting up Oracle RAC required networks.
- ▶ Setting up the ASM storage volumes.

Setup PowerVC image for Oracle RAC

The PowerVC AIX image should follow these specifications:

1. It should have a rootvg disk size of at least 50 GB. The larger size is to accommodate a 16 GB paging space and 8 GB for */tmp* to be used as the Ansible temp directory. This rootvg disk is expected to be *hdisk0* in each node.
2. A second disk of 75 GB that is not assigned to a volume group. It will be used by the installer to create *oravg* volume group which hosts Oracle Grid HOME directory. It is expected to be *hdisk1* in the nodes. Note that ***ofa_fs** variable in *vars/powervc_rac.yml* file is set to 73G based on this disk size being 75 GB. If you capture and image with a larger sized disk, then you may want to update the value of that variable accordingly.

Note: While the *hdisk0* and *hdisk1* are expected to be deployed by PowerVC as *hdisk0* and *hdisk1*, the requirement is that each one of them would have the same *hdisk* number in both nodes. However, they do not have to be *hdisk0* and *hdisk1* respectively.

3. The OS installed on the image should be AIX 7.2 TL4 SP1 or newer or AIX 7.3.
4. The following filesets need to be installed on the AIX version prior to the process of installing Oracle RAC software. While they may be installed in the nodes after they are created, the process becomes much simpler if they are installed in the source LPAR before capturing it as the PowerVC image:

- bos.adt.base
- bos.adt.lib
- bos.adt.libm
- bos.perf.libperfstat
- bos.perf.perfstat
- bos.perf.proctools
- bos.loc.utf.EN_US
- bos.rte.security
- bos.rte.bind_cmds
- bos.compat.libs
- xIC.aix61.rte
- xIC.rte
- rsct.basic.rte
- rsct.compat.clients.rte
- xlsmp.msg.EN_US.rte
- xlf rte.aix61
- openssh.base.client
- expect.base
- perl.rte
- Java8_64.jre
- dsm

5. The unzip RPM should be installed on the image. You can download the latest version from [here](#).

6. Update the image section in the `vars/powervc.yml` file in the collection with the **image**, **image_aix_version** and **image_password** with the latter set to the AIX root password value.

Setup networks for Oracle RAC

A 2-node Oracle RAC cluster has the following network requirements:

1. The two nodes require nine IP addresses, four IP addresses per node and one for the RAC scan service. These nine IP addresses must be catered by three different networks.

Note: While the Oracle RAC collection can support up to an 8-node cluster, it has been extensively tested for a 2-node cluster. If you intend to use the collection to provision a cluster with more than two nodes, then all variables files must be reviewed to update the variables' values accordingly.

Also note that while this collection documents using a single IP address for RAC scan service, this is for testing purpose only. Oracle recommends that you use three IP addresses for this service.

2. The four IP addresses per node are as follows:
 - a. One public IP address for the node's external access, sourced from the first network,
 - b. Two private IP addresses, sourced from the second and third networks respectively, for Oracle RAC private interconnect,
 - c. One virtual IP address per node.
3. Five of these nine IP addresses are sourced from network 1 and must support a 2-way resolution against a DNS server.
4. Network one must be accessible from the Ansible server because the public IP addresses or their hostnames are used as Ansible inventory entries.
5. All three networks must be routed via the VIOS and added to PowerVC.
6. All network interfaces must be consistent across the nodes. For instance, if the public IP address for node 1 is set to en0, then node2 must also use en0 for its public IP address. Table 4-1 details the requirements for the IP addresses on a two node cluster.

Note: When using PowerVC for provisioning the nodes, ensure that both **nodeX_net_ports** variables in `vars/powervc.yml` file list the public port, private 1 and virtual 2 ports in this sequence which will guarantee using en0, en1 and en2 respectively.

Table 4-1 IP addresses specifications for a 2-node Oracle RAC

Location	Function	Network #	Type (Interface)	DNS required?
Node1	Public IP	Network 1	Physical (en0)	Yes
Node2	Public IP	Network 1	Physical (en0)	Yes
Node1	First private IP	Network 2	Physical (en1)	No
Node2	First private IP	Network 2	Physical (en1)	No
Node1	Second private IP	Network 3	Physical (en2)	No
Node2	Second private IP	Network 3	Physical (en2)	No
Node1	Virtual IP	Network 1	Virtual (N/A)	Yes
Node2	Virtual IP	Network 1	Virtual (N/A)	Yes
Cluster	Cluster scan IP	Network 1	Virtual (N/A)	Yes

7. Update the network section of the *vars/powervc.yml* file, supplying the network names and IP addresses in the variables named according to the function defined in Table 4-1 on page 197 above.
8. Also update the *vars/powervc.yml* file with the following additional variables:
 - DNS server and domain.
The DNS server should be capable of forward and reverse name resolution for all 5 IP addresses labeled Yes in the last column of Table 4-1 on page 197.
 - NTP server.
The name server is needed to keep the cluster operational. Alternatively, you will need to ensure that the date and time are synchronized between the 2 nodes.
 - NFS server and its export directory as well as the directory to be used as NFS mount point in the nodes.
Oracle RAC installation binaries including the grid, database, OPatch and RU should be stored in subdirectories under that export directory in the NFS server. Check to see if a node deployed from the PowerVC image would be able to successfully mount the export directory from the NFS server.

Define the shared storage for Oracle RAC

Oracle RAC uses ASM (Automatic Storage Management) disks that are shared among the RAC nodes. The RAC collection creates four ASM diskgroups using ASM shared disks. Each diskgroup can have one or more disks.

Disks of each diskgroup should be of the same size, characteristics and similar IO speed. Namely, these four diskgroups are:

1. The OCRVOTE diskgroup stores OCR (Oracle Cluster Registry) and Voting disks information.
2. The GIMR (Grid Infrastructure Management Recovery) diskgroup contains a multi-tenant database (MGMTDB) with one pluggable database.
3. The ACFS (ASM Cluster File System) diskgroup is used for staging the Oracle database home binaries.
4. The DATA diskgroup is used for staging the database files.

The Oracle RAC installer expects each disk to have the same hdisk number in all of the RAC nodes.

Note: When using the PowerVC playbook from the collection to provision the nodes, it guarantees that each has the same hdisk number across all nodes. It does so by creating them one at a time and after creating each one, it attaches it to all nodes and then runs **cfgmgr** to ensure it captured the next available sequential number for the hdisk in all nodes before moving on to the next disk.

Table 4-2 on page 199 cross matches the above diskgroups and their corresponding variable names, disks count and each disk's size as set as default in the **disks** list of the *vars/powervc.yml* file. It also shows the corresponding hdisk number as set in the **diskgroups** variable in the *vars/powervc_rac.yml* file.

Table 4-2 *diskgroups names and corresponding variables in vars/powervc.yml*

Diskgroup	Variable name	Disks count	hdisk number	Disk size
OCRVOTE	"{{racName}}-ASMOCRx"	4	2 3 4 5	10
GIMR	"{{racName}}-GIMRx"	2	6 7	40
ACFS	"{{racName}}-ACFS-DBHome"	1	8	75
DATA	"{{racName}}-DBDiskx"	2	9 10	10

Keep in mind the following considerations as you update these variables in their variables file:

- ▶ The **diskgroups** variable in the *vars/powervc_rac.yml* file lists the hdisk number of these disks. With a PowerVC nodes' deployment, the image has taken hdisk0 and hdisk1 as described in "Setup PowerVC image for Oracle RAC" on page 196, consequently hdisk number of each of these ASM disks would be as shown in the 'hdisk number' column in Table 4-2 above. If you change the count of any of the diskgroups then you will need to update the hdisk numbers in that **diskgroups** variable in the *vars/powervc_rac.yml* file.
- ▶ You may change the disks' sizes to meet your requirements. Ensure that all disks of a given diskgroup have the same size.
- ▶ The **vol_size_GB** variable in *vars/powervc_rac.yml* is set to 75 GB based on the ACFS disk size being 75 GB. If you change that disk's size in *vars/powervc.yml* then update that variable in *vars/powervc_rac.yml* file as well.

Note: When using PowerVC playbook from the collection to provision the nodes, it guarantees that all non-rootvg disks' headers are clear and has no PVID because it carves them fresh from the storage subsystem.

For manual setup of the nodes, use `chdev -l hdiskX -a pv=clear` to clear the PVID. Then use `lquerypv -h /dev/hdiskX` to check if the header is clear, If it is not, then use `dd if=/dev/zero of=/dev/hdiskX bs=1024k count=100` to clear it.

The collection roles used for infrastructure provisioning

The *power_aix_oracle_rac_asm* collection contains the following roles pertaining to the infrastructure layer provisioning:

- ▶ **powervc_create_network_ports**
This role creates an OpenStack port to be used during the node creation. A port defines the IP address and network to be used for a given interface.
- ▶ **powervc_create_nodes_without_rac_volumes**
This role uses parameters set in the *vars/powervc.yml* file to create the new cluster nodes.
- ▶ **powervc_obtain_token**
This role obtains a PowerVC access token to establish a REST API connection from Ansible server to PowerVC server. The subsequent ASM disks creation role requires REST API access, hence the need for the token.
- ▶ **powervc_create_and_multiattach_asm_volumes**
This role creates the ASM disks one at a time. Upon creating each disk, this role attaches it to all nodes and runs `cfgmgr` to ensure the disk maintains the same hdisk number in all nodes as required by Oracle RAC installer.

► **powervc_add_nodes_to_inventory**

This role updates the inventory file with the nodes and additional parameters to set it up for Ansible management and the prepares the environment for execution of the second playbook that is responsible for grid and database software installation.

Automate Oracle RAC software installation

The Oracle RAC software installation automation involves the activities which are automated via *install_and_configure_Oracle_RAC.yml* playbook and its supported Ansible roles which are provided by the collection. Oracle database installation via the collection assumes *software-installation only* option on the ACFS shared filesystem.

When you build the infrastructure manually, installation of the grid and database software requires updating *vars/rac.yml* file with the values of the required variables and include only *vars/rac.yml* file in the playbook. If you automate the infrastructure provisioning with PowerVC as per previous section, then you would have populated those variables in the *vars/powervc.yml* file, which are then referenced in *vars/powervc_rac.yml* file. In this case you would need to include both *vars/powervc.yml* file and *vars/powervc_rac.yml* file in the playbook.

The collection roles used for Oracle RAC software installation

The `power_aix_oracle_rac_asm` collection contains the following roles pertaining to the grid and database software installation:

► **bootstrap**

This role sets up the basic environment to enable full functionality of Ansible, setting the nameserver, binding and password-less connections to the RAC nodes.

► **preconfig**

This role sets up basic environment such as time of day, configuration for accessing the Internet and ensuring consistent AIX version, release, TL, and SP. It also NFS mounts and installs the AIX filesets.

► **config**

This role sets up AIX to meet the requirements for installing Oracle RAC software.

► **install**

This role creates the ASM disk groups and ACFS, prepares for installing Grid and database, and finally installs GRID_HOME on a JFS2 filesystem and Database ORACLE_HOME on the ACFS shared Filesystem.

Installation steps for the different options

There are 2 options for using the collection to install Oracle RAC on AIX:

- Option 1: Manually configure infrastructure while automating the software installation,
- Option 2: Fully automate both operations via a single playbook run.

Option 1 is shown in Figure 4-4 on page 193. Option 2 is shown in Figure 4-5 on page 194

Option 1 installation steps

Installation option 1 entails preparing the cluster nodes manually and then using the collection to install the RAC software.

1. Create the Oracle RAC cluster AIX nodes manually.
2. Create the local and shared storage volumes manually as per the specifications and attach them to the nodes.
3. Ensure the nodes are configured with the correct networks per the specifications.

4. Ensure the hostnames, virtual and scan IP addresses are added correctly to the DNS server.
5. If no NTP server is configured for the nodes, then ensure the clock on the cluster nodes are in sync.
6. NFS servers are needed for the AIX filesets installation as well as for staging the Oracle 19c software.
7. Update *vars/rac.yml* file to reflect the correct values for all the concerned variables.
8. Review the *inventory* and *ansible.cfg* files and ensure the nodes are added correctly to the *inventory* file with the correct name of the nodes and the group containing them.
9. Update the *install_and_configure_Oracle_RAC.yml* playbook shown in Example 4-5 below as follows:
 - a. Uncomment the **hosts:** line and set the field by specifying the inventory group name per step 8.
 - b. Uncomment the first variables file (named *vars/rac.yml*) to have its variables included in this execution.
10. Run the playbook as follows: **ansible-playbook install_and_configure_Oracle_RAC.yml** the output is shown in Example 4-5.

Example 4-5 Contents of install_and_configure_Oracle_RAC.yml playbook

```

---
# install_and_configure_Oracle_RAC.yml
- hosts: "{{ racName }}"
  gather_facts: no
  vars_files:
#   - vars/rac.yml
#   - vars/powervc.yml
#   - vars/powervc_rac.yml
  roles:
    - role: bootstrap
      vars:
        download_dir: "~"
        target_dir: "/tmp/.ansible.cpdire"
      tags: bootstrap
    - role: preconfig
      tags: preconfig
    - role: config
      tags: config
    - role: install
      tags: install

```

Option 2 Installation assumptions

The following assumption apply to installation option 2:

- ▶ A variable named **racName** is used throughout the automation process and is a required as an extra parameter for playbooks of both operations. The following parameters inherit the **racName** variable:
 - a. Nodes' names and hostnames append a counter to the **racName**. So, if the **racName** is set to *myorac* then the nodes would be named *myorac1* and *myorac2*.
 - b. The nodes' group on the Ansible inventory file is named after the **racName**.
 - c. Consequently, the target hosts in the software installation playbook is set to that Ansible inventory group which is named **racName**.
 - d. The hostname of the cluster scan IP address appends '-scan' to the **racName**.
 - e. The hostnames of all node IP addresses are based on the nodes' names which are based on the **racName**.

Troubleshooting Note: If any of the nodes is rebuilt (or otherwise its ssh identity has changed) after been added to Ansible server's `~/.ssh/known_hosts`, then entries in that file need to be cleared out prior to any subsequent attempt to install Oracle RAC software in these nodes. This is true for installation option 1 as well.

This precautionary step would prevent running into a **“WARNING: POSSIBLE DNS SPOOFING DETECTED!”** which would cause password-less ssh connection to fail and subsequently failure of the playbook execution on node1 and therefore incomplete software installation process.

Option 2 Installation steps

Installation option 2 entails using the collection to automate both preparing the infrastructure layer (operation 1), and installing the RAC software (operation 2) in a single execution. The process invokes importing the playbook for operation 2 as the last step of operation 1's playbook in order to automate executing them both sequentially.

1. Create DNS entries in the DNS server for the 5 addresses that are required to have DNS entries per Table 4-1 on page 197.
2. Configure the NFS server and store the Oracle RAC software in subdirectories in its export directory. Note that the export directory value needs to be set in the `vars/powervc.yml` file and names of the binaries files and their hosting subdirectories need to be set in the `vars/powervc_rac.yml` file.
3. Update the `vars/powervc.yml` file with values for all the image, networks and shared storage disks variables as well as the DNS, NFS, NTP servers and the PowerVC server credentials as well.
4. Update `vars/powervc_rac.yml` file for the hdisk0 and ACFS disk size per “Option 1 installation steps” on page 200. Furthermore, update the grid and oracle passwords and any other variable deemed necessary for your environment.
5. Copy PowerVC certificate file from `/etc/pki/tls/certs/powervc.crt` file in the PowerVC server to the Ansible server to be used in the next step.
6. Copy the `/opt/ibm/powervc/powervcrc` file from the PowerVC server to the Ansible server, update its `OS_CACERT` to the location where you copied the PowerVC certificate file. Also update it with the user id and password of the PowerVC server and source it.
7. Example 7 shows the playbook for infrastructure provisioning layer. Contents of `powervc_build_AIX_RAC_nodes.yml` playbook

```

---
# powervc_build_AIX_RAC_nodes.yml

- name: Build and configure the RAC nodes using PowerVC
  hosts: localhost
  gather_facts: no
  vars_files:
    - vars/powervc.yml
  tasks:
    - include_vars: "vars/powervc.yml"
    - fail:
      msg: "racName is required for this playbook to build a dual-node Oracle RAC."
      when: racName is not defined

- name: Display the input name prefix and count of VMs to be built
  debug:
    msg: "Creating nodes {{racName}}1 and {{racName}}2 for this dual-node Oracle RAC."

```



```

- name: define the network ports based on the networks and IP addresses to be used.
  import_role: name=powervc_create_network_ports

- name: Create new AIX VMs to act as Oracle RAC nodes
  import_role: name=powervc_create_nodes_without_rac_volumes

- import_role: name=powervc_obtain_token
- include_role: name=powervc_create_and_multiattach_asm_volumes
  with_items: "{{ disks }}"

- name: Now the nodes are good to go, add them to the inventory file to be managed by
Ansible
  import_role: name=powervc_add_nodes_to_inventory

# Importing the playbook to be used for installing and configuring the Oracle RAC.
- import_playbook: install_and_configure_Oracle_RAC.yml

```

8. Notice that the last step in it invokes the software installation playbook which is shown in Example 4-6.

You need to update the software installation playbook by uncommenting the last variables file (namely - *vars/powervc_rac.yml*) as the variables defined in that file are needed in the playbook.

Example 4-6 Contents of install_and_configure_Oracle_RAC.yml

```

---
# install_and_configure_Oracle_RAC.yml
# Powervc based deployments uses variable files vars/powervc.yml,vars/powervc_rac.yml
# If the LPARs are build manually, to automate oracle RAC deployment use variable file
vars/rac.yml
#- hosts: "{{ racName }}" # racName variable is defined when you use the powervc
automation scripts for building the AIX LPARs
#- hosts: orac # Get the group name from inventory file which contains the
oracle cluster nodes
gather_facts: no
vars_files:
# - vars/powervc.yml
# - vars/powervc_rac.yml
# - vars/rac.yml
roles:
- role: bootstrap
  vars:
    download_dir: "~"
    target_dir: "/tmp/.ansible.cpsdir"
  tags: bootstrap
- role: preconfig
  tags: preconfig
- role: config
  tags: config
- role: install
  tags: install

```

9. Run operation 1's playbook (which will now perform both operations) as follows:

```
ansible-playbook powervc_build_AIX_RAC_nodes.yml -e racName=myorac.
```

Running the playbooks separately

You may choose to run the 2 playbooks in 2 separate steps for different reasons such as:

- ▶ A time gap is needed to prepare the DNS server and updating it with the required addresses.
- ▶ A time gap is needed to prepare the NFS server and/or setting it up with the required Oracle software binaries to be used in the second operation.
- ▶ Just to interrogate the nodes for all the prerequisites to ensure the first operation's playbook has satisfied them.

In that case, you can run the infrastructure playbook the same way as in step Example 7 on page 202 with the only exception of commenting out the last line in that playbook (shown in Example 4-6 on page 203) to avoid importing the software installation playbook.

Then when you are ready to do the software installation, you follow these two steps:

1. Uncomment the middle variables file in the software installation playbook (shown in Example 4-6 on page 203) which reads - `vars/powervc.yml` so that both the second and third variable files that have PowerVC in their name are active.
2. Run the software installation playbook with `racName` as a variable as follows:
`ansible-playbook install_and_configure_Oracle_RAC.yml -e racName=myorac`

As you continue to work with the collection, you might find it useful to refer to the [collection documentation](#) for newer release updates. Furthermore, the collection's [Github issues page](#) is a good tool to leverage for resolving any issues should they arise.

Note: Notice that the Oracle RAC automation collection stops at the software installation. To create database instances and manage them you can use section 4.6.4, “Automating Oracle DBA operations” on page 204

4.6.4 Automating Oracle DBA operations

Oracle Database Administration tasks are a set of support activities which are done by Database Administrators (DBAs). These tasks are mostly iterative and often requires a lot of attention especially when there are lot of databases to handle. Automating these tasks would help DBAs save time and avoid human errors.

The `ibm.power_aix_oracle_dba` is an Ansible collection based on an Opensource project called “Oravirt”, which automates Oracle Database administration tasks. These roles and modules have been evaluated and tested to work with Oracle databases running on AIX and offer a wide range of playbooks.

There is a video created by one of the authors of this publication that shows the use of Ansible for Day 2 operations (DBA operations such as creating a new database instance, creating a new user, and patching Oracle). It can be viewed at <https://1nkd.in/dfkB6FeS>.

Prerequisites

The following software must be installed on the Ansible controller host.

- ▶ Python 3.6 or later (Python can be installed using `dnf install python3`).
- ▶ `Cx_oracle`

This is a python module which makes the connection to the database using sys privileges. More information on `Cx_oracle` can be found [here](#).

To install `Cx_oracle` online use one of the following commands:

- As root: `python -m pip install cx_Oracle --upgrade`

- As a non-root user: `python -m pip install cx_Oracle--upgrade --user`

For an offline installation do the following:

- Download the source distribution from <https://pypi.org/project/cx-Oracle/#files> and place it a location, for example `/tmp`
- Run the command `python3 -m pip install --no-build-isolation /tmp/cx_Oracle-8.3.0.tar.gz`.

Note: If there are multiple python versions, the python version which was used to install `cx_Oracle` must be used for running the playbooks. To verify which Python version see Example 4-7 for assistance.

Example 4-7 shows how to validate which version of Python was used to install the `Cx_Oracle` package and which must be used to run the playbooks.

Example 4-7 Determine the version of Python used to install Cx_Oracle package

```
$ pip3.9 show cx_Oracle
Name: cx-Oracle
Version: 8.3.0
Summary: Python interface to Oracle
Home-page: https://oracle.github.io/python-cx_Oracle
Author: "Anthony Tuininga",
Author-email: "anthony.tuininga@gmail.com",
License: BSD License
Location: /home/ansible/local/lib/python3.9/site-packages
Requires:
Required-by:
```

As we can see from Example 4-7, the location of `Cx-Oracle` is in `python3.9 site-packages`. So, `python3.9` must be used as the python interpreter to run the playbooks.

Getting Started

To use the Oracle DBA operations collection, Ansible version 2.9 or later must be installed on RHEL 8.x or later for either Linux on Power or x86-64. You can download this collection from any of the following public repositories:

- Galaxy: https://galaxy.ansible.com/ui/repo/published/ibm/power_aix_oracle_dba/
- Github: <https://github.com/IBM/ansible-power-aix-oracle-dba>

1. Install the collection using this command:

```
$ ansible-galaxy collection install ibm.power_aix_oracle_dba
```

2. Download and extract the Oracle Instant client software from [Oracle site](#). When you arrive at the download site click on “other platforms” as shown in Figure 4-7 on page 206 to get the option for downloading the Linux on Power client.

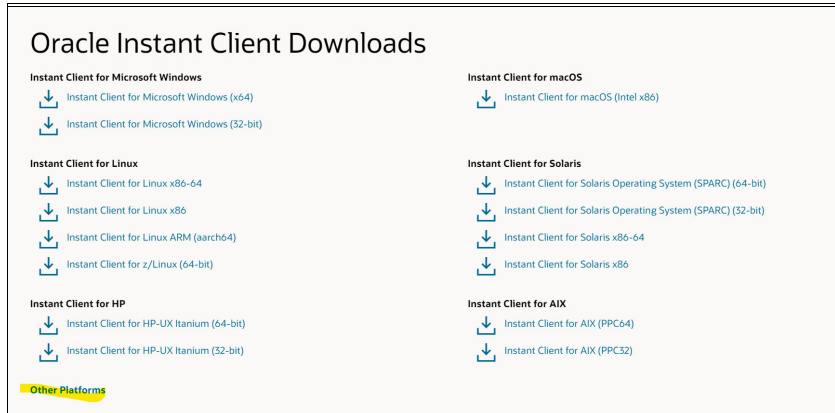


Figure 4-7 Oracle Instant Client Download initial page

3. On the next page, select Linux on Power Little Endian as shown in Figure 4-8.

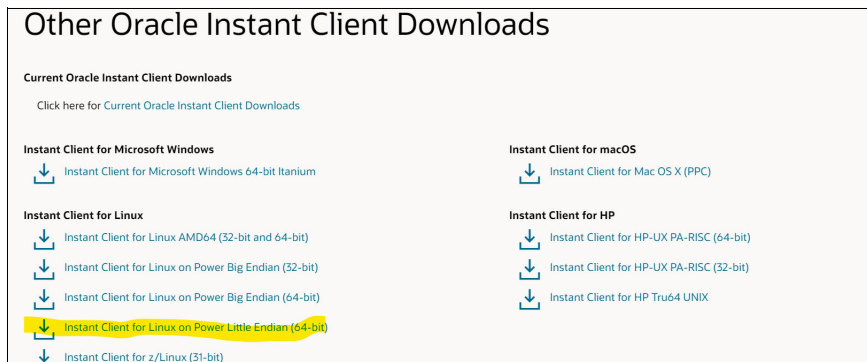


Figure 4-8 Other Oracle Instant Client downloads

4. Install the packages “libnsl” & “libaio” using yum/dnf.

```
dnf install libnsl -y
dnf install libaio -y
```

5. Prepare the inventory file with the hostnames of the virtual machines where the Oracle databases are running.

Running admin tasks

In this section we'll provide detailed steps on running two different admin tasks:

- ▶ Create a Database
- ▶ Manage Database Users

Create Database

The role “oradb_create” is used to create databases. It can be used for a Non Container Database (CDB) instance or a CDB in a Single Instance or RAC. In this example we're going to create a RAC Container Database (CDB) called “devdb” with one PDB called “devpdb”.

To create the database follow these steps:

1. Passwordless SSH must be established between Ansible user & Oracle Database user.
2. Define the required hostname in an inventory file to be used to execute the playbook.
3. There are three files which needs to be updated:
 - a. {{ collection_dir }}/power_aix_oracle_dba/playbooks/vars/vault.yml: This contains the SYS user password of ASM and SYS password which needs to set to the new database.

- b. `{{ collection_dir }}/power_aix_oracle_dba/playbooks/create-db.yml`: This contains the playbook which executes the “`oradb_create`” role.
- c. `{{ collection_dir }}/power_aix_oracle_dba/playbooks/vars/create-db-vars.yml`: This contains all the required variables required to create a database. Multiple databases can be created by providing the variables as a list.
4. Update the `{{ collection_dir }}/power_aix_oracle_dba/playbooks/vars/vault.yml` file with the passwords.
 - a. Go to the playbooks directory and update the file with system password for `asm` and `dba` as shown in Example 4-8.

Example 4-8 Update the passwords

```
$ cat vault.yml
default_gipass: Oracle4u
default_dbpass: Oracle4u
```

- b. Encrypt the file using the command:

```
$ ansible-vault encrypt vault.yml
```

5. Update the `hosts` and `remote_user` in the directory:

```
{{ collection_dir }}/power_aix_oracle_dba/playbooks/create-db.yml file
```

This is shown in Example 4-9.

Example 4-9 Modify create-db.yml

```
- name: Create a Database
  hosts: rac91                # Target LPAR hostname defined in the inventory file.
  remote_user: oracle        # Oracle Database Username
  vars_file:
    - vars/create-db-vars.yml
    - vars/vault.yml
  roles:
    - { role: ibm.power_aix_oracle_dba.oradb_create }
```

6. Update the following variables in

```
{ collection_dir }}/power_aix_oracle_dba/playbooks/vars/create-db-vars.yml
```

This is shown in Example 4-10.

Example 4-10 Variables in create-db-vars.yml

```
oracle_stage: /tmp # Location on the target AIX LPAR to stage response files.
oracle_inventory_loc: /u01/app/oraInventory
oracle_base: /u01/base
oracle_dbf_dir_asm: '+DATA1' # If storage_type=ASM this is where the database is placed.
oracle_reco_dir_asm: '+DATA1' # If storage_type=ASM this is where the fast recovery area is
oracle_databases:      # Dictionary describing the databases to be created.
  - home: db1
    oracle_version_db: 19.3.0.0 # For a 19c database, the version should be 19.3.0.0
    oracle_home: /u01/app/19c_ansi # Oracle Home location.
    oracle_edition: EE # The edition of database-server (EE,SE,SEONE)
    oracle_db_name: devdb # Database name
    oracle_db_type: RAC # Type of database (RAC,RACONENODE,SI)
    is_container: True # (true/false) Is the database a container database.
    pdb_prefix: devpdb # Pluggable database name.
    num_pdb: 1 # Number of pluggable databases.
    storage_type: ASM # Database storage to be used. ASM or FS.
    service_name: db19c # Initial service to be created.
    oracle_init_params: "" # initialization parameters, comma separated
```

```

oracle_db_mem_totalmb: 10000 # Amount of RAM to be used for SGA + PGA
oracle_database_type: MULTIPURPOSE # MULTIPURPOSE|DATA_WAREHOUSING|OLTP
redolog_size_in_mb: 512 # Redolog size in MB
state: present # present | absent

```

7. Verify the DB does not currently exist as shown in Example 4-11.

Example 4-11 Verifying DB does not exist

```

bash-5.1$ srvctl status database -d devdb
PRCD-1120 : The resource for database devdb could not be found.
PRCR-1001 : Resource ora.devdb.db does not exist
bash-5.1$

```

8. Execute the following command to run the playbook as shown in Example 4-12.

Example 4-12 Output from running the playbook

```

[ansible@x134vm236 playbooks]$ ansible-playbook create-db.yml -i inventory.yml
--ask-vault-pass
Vault password:
[DEPRECATION WARNING]: "include" is deprecated, use include_tasks/import_tasks instead.
This feature will be removed in version 2.16.
Deprecation warnings can be disabled by setting deprecation_warnings=False in ansible.cfg.

PLAY [Create a Database]
*****

TASK [Gathering Facts]
*****
[WARNING]: Platform aix on host rac93 is using the discovered Python interpreter at
/usr/bin/python3, but future installation of
another Python interpreter could change the meaning of that path. See
https://docs.ansible.com/ansible-
core/2.12/reference_appendices/interpreter_discovery.html for more information.
ok: [rac93]

TASK [oradb_create : set fact]
*****
ok: [rac93] => (item={'home': 'db1', 'oracle_version_db': '19.3.0.0', 'oracle_home':
'/u01/app/oracle/db', 'oracle_edition': 'EE', 'oracle_db_name': 'devdb', 'oracle_db_type':
'RAC', 'is_container': True, 'pdb_prefix': 'devpdb', 'num_pdb': 1, 'storage_type': 'ASM',
'service_name': 'devdb', 'oracle_init_params': '', 'oracle_db_mem_totalmb': 10000,
'oracle_database_type': 'MULTIPURPOSE', 'redo_log_size_in_mb': 50, 'state': 'present'})

TASK [oradb_create : Create Stage directory for response file.]
*****
ok: [rac93]

TASK [oradb_create : listener | Create responsefile for listener configuration]
*****
skipping: [rac93] => (item={'home': 'db1', 'oracle_version_db': '19.3.0.0', 'oracle_home':
'/u01/app/oracle/db', 'oracle_edition': 'EE', 'oracle_db_name': 'devdb', 'oracle_db_type':
'RAC', 'is_container': True, 'pdb_prefix': 'devpdb', 'num_pdb': 1, 'storage_type': 'ASM',
'service_name': 'devdb', 'oracle_init_params': '', 'oracle_db_mem_totalmb': 10000,
'oracle_database_type': 'MULTIPURPOSE', 'redo_log_size_in_mb': 50, 'state': 'present'})

NOTE: Some output has been truncated.

TASK [oradb_create : Add dotprofile (2)]
*****

```

```

changed: [rac93] => (item=[{'home': 'db1', 'oracle_version_db': '19.3.0.0', 'oracle_home':
'/u01/app/oracle/db', 'oracle_edition': 'EE', 'oracle_db_name': 'devdb', 'oracle_db_type':
'RAC', 'is_container': True, 'pdb_prefix': 'devpdb', 'num_pdb': 1, 'storage_type': 'ASM',
'service_name': 'devdb', 'oracle_init_params': '', 'oracle_db_mem_totalmb': 10000,
'oracle_database_type': 'MULTIPURPOSE', 'redolog_size_in_mb': 50, 'state': 'present'},
{'changed': False, 'stdout': '', 'stderr': '', 'rc': 0, 'cmd': 'ps -ef | grep -w
"ora_pmon_devdb" | grep -v grep | sed \'s/^.*pmon_//g\'', 'start': '2023-09-06
02:09:57.933677', 'end': '2023-09-06 02:09:57.983788', 'delta': '0:00:00.050111', 'msg':
'', 'invocation': {'module_args': {'_raw_params': 'ps -ef | grep -w "ora_pmon_devdb" | grep
-v grep | sed \'s/^.*pmon_//g\'', '_uses_shell': True, 'warn': False, 'stdin_add_newline':
True, 'strip_empty_ends': True, 'argv': None, 'chdir': None, 'executable': None, 'creates':
None, 'removes': None, 'stdin': None}}, 'stdout_lines': [], 'stderr_lines': [], 'failed':
False, 'item': {'home': 'db1', 'oracle_version_db': '19.3.0.0', 'oracle_home':
'/u01/app/oracle/db', 'oracle_edition': 'EE', 'oracle_db_name': 'devdb', 'oracle_db_type':
'RAC', 'is_container': True, 'pdb_prefix': 'devpdb', 'num_pdb': 1, 'storage_type': 'ASM',
'service_name': 'devdb', 'oracle_init_params': '', 'oracle_db_mem_totalmb': 10000,
'oracle_database_type': 'MULTIPURPOSE', 'redolog_size_in_mb': 50, 'state': 'present'},
'ansible_loop_var': 'item']])

```

```
TASK [oradb_create : Check if database is running]
```

```
*****
changed: [rac93]
```

```
TASK [oradb_create : debug]
```

```
*****
ok: [rac93] => {
  "psout.stdout_lines": [
    "   grid 14483936      1   0 00:54:37      -   0:00 asm_pmon_+ASM1",
    "   grid 14745992      1   0 00:54:56      -   0:00 apx_pmon_+APX1",
    "   oracle 21365224    1   0 02:08:59      -   0:00 ora_pmon_devdb1"
  ]
}

```

```
PLAY RECAP
```

```
*****
rac93                : ok=11   changed=4   unreachable=0   failed=0   skipped=3
rescued=0   ignored=0

```

9. Verify the DB is created by running the commands shown in Example 4-13.

Example 4-13 Verify database is created

```

bash-5.1$ srvctl status database -d devdb
Instance devdb1 is running on node rac93
Instance devdb2 is running on node rac94

```

Manage Database Users

The role “oradb_manage_users” is used to create, drop, lock, unlock & set expiration to database users. It uses the “oracle_users” module. The users require privileges to access the database which can be achieved by the role “oradb_manage_grants”. It uses the “oracle_grants” module.

In the following example we're going to create two database users (testuser1 & testuser2) in a pluggable database DEVPDB running in a container database and grant privileges to the users.

1. There are two files which need to be updated:
 - a. {{ collection_dir }}/power_aix_oracle_dba/playbooks/vars/manage-users-vars.yml: This contains database hostname, database port number and the path to the Oracle client.

- b. `{{ collection_dir }}/power_aix_oracle_dba/playbooks/vars/vault.yml`: This contains sys password which will be used by `cx_oracle` to connect to the database with `sysdba` privilege.
2. Update the common variables file `{{collection_dir}}/power_aix_oracle_dba/playbooks/vars/manage-users-vars.yml` as shown in Example 4-14.

Example 4-14 Update the common variables file

```
# Create/Drop Database Users - Variables section
hostname: rac93 # AIX Lpar hostname where the database is running.
listener_port: 1522 # Database port number.
oracle_db_home: /home/ansible/oracle_client # Oracle Instant Client path on controller.
oracle_databases: # Database users list to be created
  - users:
    - schema: testuser1 # Username to be created.
      default_tablespace: users # Default tablespace to be assigned to the user.
      service_name: devpdb # Database service name.
      schema_password: oracle3 # Password for the user.
      grants_mode: enforce # enforce|append.
      grants:
        - connect # Provide name of the privilege as a list to
grant to the user.
        - resource
      state: present # present|absent|locked|unlocked [present: Creates user,
# absent: Drops user]
# Multiple users can be created with different attributes as shown below.
- users:
  - schema: testuser2
    default_tablespace: users
    service_name: devpdb
    schema_password: oracle4
    grants_mode: enforce
    grants:
      - connect
      state: present # present|absent|locked|unlocked [present: Creates user,
# absent: drops user}
```

3. Update the passwords file:

`{{ collection_dir }}/power_aix_oracle_dba/playbooks/vars/vault.yml`

with sys user password. This file needs to be encrypted using `ansible-vault`. While running the playbook, please provide the vault password. This is shown in Example 4-15.

Example 4-15 Update the passwords file

```
default_dbpass: Oracle4u # SYS password
default_gipass: Oracle4u # ASMSNMP password
```

4. Encrypt the passwords file using `ansible-vault` as shown in Example 4-16

Example 4-16 Encrypt passwords file

```
$ ansible-vault encrypt vars/vault.yml
New Vault password:
Confirm New Vault password:
Encryption successful
```

5. Check the user names in the database before creating them as shown in Example 4-17 on page 211.

Example 4-17 Validate user names in the database

```
SQL> sho pdbs

      CON_ID CON_NAME                OPEN MODE  RESTRICTED
-----
          3  DEVPDB                READ WRITE NO
SQL> select username from dba_users where username in ('TESTUSER1','TESTUSER2');

no rows selected
```

The output shows that the users do not exist.

6. Create the playbook from {{ collection_dir }}/power_aix_oracle_dba/playbooks directory. This is shown in Example 4-18.

Example 4-18 Create manage-users playbook

```
$ cat manage-users.yml
- hosts: localhost
  connection: local
  gather_facts: false
  vars_files:
    - vars/manage-user-vars.yml
    - vars/vault.yml
  roles:
    - { role: oradb_manage_users }
```

7. Now execute the playbook as shown in Example 4-19.

Example 4-19 Execute the playbook to create users

```
[ansible@x134vm236 playbooks]$ ansible-playbook manage-users.yml --ask-vault-pass
Vault password:

PLAY [Create DB User]
*****
*****

TASK [oradb_manage_users : Manage users (cdb/pdb)]
*****
changed: [localhost] => (item=port: 1522 service: devpdb schema: testuser1 state:present)
changed: [localhost] => (item=port: 1522 service: devpdb schema: testuser2 state:present)
[WARNING]: Module did not set no_log for update_password

PLAY RECAP
*****
*****
localhost                : ok=1    changed=1    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
```

8. Check the user names in the database after creating them as shown in Example 4-20 where we can see the testuser1 & testuser2 are created in the PDB database.

Example 4-20 Display user names in the database

```
SQL> sho pdbs

      CON_ID CON_NAME                OPEN MODE  RESTRICTED
-----
          3  DEVPDB                READ WRITE NO
SQL> select username from dba_users where username in ('TESTUSER1','TESTUSER2');
```

USERNAME

TESTUSER2

TESTUSER1

To run other playbooks please refer the readme files in [ansible-power-aix-oracle-dba/docs](https://github.com/ansible-power-aix-oracle-dba/docs) for each corresponding DB admin task.

4.7 SAP automation

If your organization relies on SAP HANA and S/4HANA for its critical business operations, downtime is not an option. Yet the complexity of manually deploying and managing an SAP environment – either on premise or in the cloud – is time consuming and error prone and can lead to service degradation, increased security exposure, and outages. Automation can help by:

- Streamlining repetitive SAP management tasks.
- Ensuring consistent configuration across systems.
- Reducing deployment time lines and unplanned downtime.

Figure 4-9 helps you understand how automation can help you over the life cycle of your SAP environments.

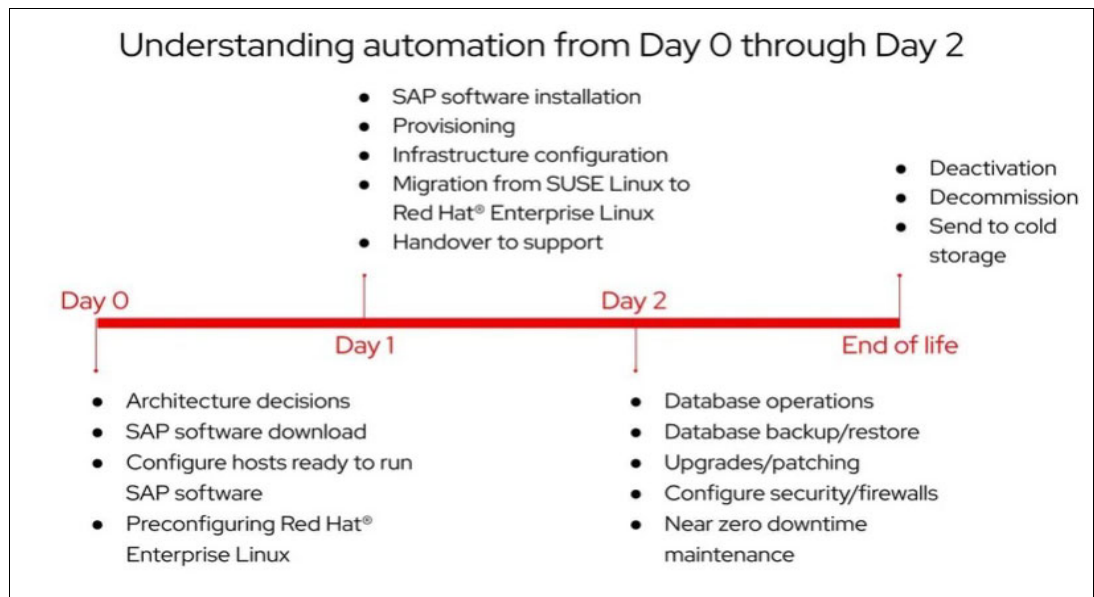


Figure 4-9 Automation benefits in your SAP environments⁵

When looking at the end-to-end SAP S/4HANA installation process, we can breakdown the process into four major blocks:

1. Server provisioning

This is the most variable block, dependent on the infrastructure. This can be done using Ansible alone or in concert with other tools such as Terraform. As SAP can be installed across a wide variety of infrastructure and cloud environments, the server provisioning is highly dependent on the infrastructure chosen for the SAP environment.

⁵ <https://catalog.redhat.com/solutions/detail/e95cc4e4b41347639b8f5da129f588ac>

2. Basic OS Setup

SAP has spent a lot of effort in understanding how to install the base operating system for the servers running the different SAP components. For SAP HANA, the operating system is Linux, either SUSE or RHEL, and there are specific documented settings defined in multiple SAP notes. Likewise, there are documented settings for NetWeaver installations.

3. HANA Installation and Configuration

4. S/4 installation and configuration

Figure 4-10 illustrates the use of Ansible for SAP installation.

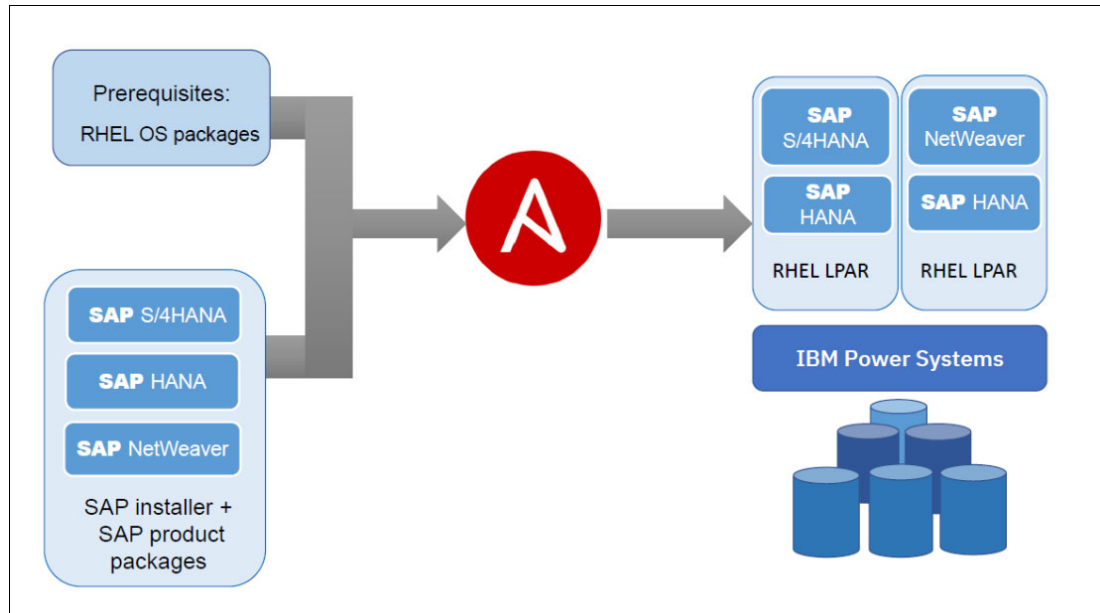


Figure 4-10 Ansible automation for SAP installation⁶

SAP and Red Hat automation options

SAP and Red Hat have partnered for more than two decades working together to provide new innovative solutions for SAP users. For SAP automation with Ansible, there are currently two different options:

► Red Hat Enterprise Linux System Roles for SAP

The [Red Hat Enterprise Linux System Roles for SAP](#) are a subset of the Red Hat Enterprise Linux System Roles which is a collection of Ansible roles and modules provided as part of your Red Hat subscription. These system roles provide a stable and consistent configuration interface to automate and manage system functions across multiple releases of Red Hat Enterprise Linux. These roles are designed to be executed by Ansible to assist administrators with server configuration right after the servers have been installed. The system roles are available either from the `rhel-system-roles` RPM or from [Red Hat Automation Hub](#).

The roles are based on development of the Linux System Roles upstream project, and for the SAP related roles, the SAP LinuxLab upstream project. These roles are fully supported by Red Hat.

► SAP LinuxLab Automation for SAP

⁶ <https://community.sap.com/t5/technology-blogs-by-sap/automating-the-installation-of-sap-s-4hana-and-sap-hana-on-ibm-power/ba-p/13462363>

The [SAP LinuxLab open-source initiative](#) provides easier creation and management of SAP environments using code and tools created by SAP Technology Partners. All capabilities available through the SAP LinuxLab open-source initiative are available for SAP Customers and SAP Service Partners. The projects within this initiative help SAP technical administrators and infrastructure administrators to run their SAP systems, by providing:

- Automation of SAP Landscapes infrastructure, OS preparation and SAP software installation with homogeneous architectures
- Automation of common operational tasks
- Sizing tools and architectural guidance for SAP Landscapes
- Other technical deliverables and tools

These projects are open source and community supported and may have more up to date content compared to the Red Hat Enterprise Linux System Roles for SAP as it takes some time to integrate new content from this GitHub project into the supported Red Hat product.

The SAP LinuxLab has a wide variety projects and tools but we are focused in this publication on those projects that assist in automation of tasks in the SAP environment including installation or Day 0 operations as well as Day 1 and Day 2 operations. The current list of SAP LinuxLab projects relevant to automation is provided in Table 4-3.

Table 4-3 SAP LinuxLab projects

Project Repository	Project Description
community.sap_install	Collection of Ansible Roles for various SAP software installation
community.sap_operations	Collection of Ansible Roles for various operational tasks with SAP Systems
community.sap_launchpad	Collection of Ansible Roles and Ansible Modules for various tasks using SAP Launchpad APIs
community.sles-for-sap	Collection of Ansible roles for SLES for SAP
terraform.templates_for_sap	Terraform Templates for deployment of various SAP solution scenarios, for every Cloud and Hypervisor
terraform.modules_for_sap	Terraform Modules for each Cloud and Hypervisor and dynamic Ansible Playbooks for SAP installations. Subcomponent of the Terraform Templates for SAP
demo.sap_install	Demonstration usage of community.sap_install collection in Ansible Automation Platform or AWX

Both of these options are described more completely in the following sections.

4.7.1 Red Hat Enterprise Linux System Roles for SAP

The Red Hat System Roles for SAP was initially introduced in RHEL 7 and is provided in the RHEL for SAP Solutions subscription and can be used by Ansible to manage RHEL systems. With system roles, you can:

- ▶ Provision and configure infrastructure and network components, OS, and applications—according to SAP HANA and SAP S/4HANA requirements.
- ▶ Manage configurations for Red Hat Enterprise Linux, SAP, and any other system that integrates with your SAP deployment.

- ▶ Standardize infrastructure provisioning and configuration processes across your environment.
- ▶ Automate error-prone manual tasks to improve accuracy and reliability.
- ▶ Patch and perform other security-related operations.

The Red Hat Enterprise Linux subscription provides support for RHEL System Roles. The roles provided by the System Roles for SAP are:

- `sap_general_preconfigure` (was named `sap-preconfigure` in earlier versions)
- `sap_netweaver_preconfigure` (was named `sap-netweaver-preconfigure` previously)
- `sap_hana_preconfigure` (was named `sap-hana-preconfigure` previously)

The RHEL System Roles for SAP, just like the RHEL System Roles, are installed and run from a central node or control node. The control node connects to one or more managed nodes and performs installation and configuration steps on them. It is recommended that you use the latest major release of RHEL on the control node (RHEL 8) and use the latest version of the roles either from the `rhel-system-roles-sap` RPM or from Red Hat Automation Hub. The RHEL System Roles for SAP and Ansible packages do not need to be installed on the systems that are being managed. Table 4-4 shows the supported combinations of managed systems and control nodes for the current version of the Linux System Roles for SAP.

Table 4-4 Supported configurations for System Roles for SAP

Control Node	Managed Node	Support Status
RHEL 8.4 or later	RHEL 8.0 or later	fully supported
RHEL 8.4 or later	RHEL 7.6 or later	fully supported
RHEL 8.4 or later	RHEL 7.5 or earlier	not supported
RHEL 8.3 or earlier	RHEL (any release)	not supported

Note: For control nodes running RHEL 7.8, RHEL 7.9, or RHEL 8.1, you can use the previous versions of `rhel-system-roles-sap` which are in Tech Preview support status. Please find the instructions for these versions [here](#).

For control nodes running RHEL 8.2 or RHEL 8.3, you can use version 2 of `rhel-system-roles-sap` which is fully supported. Please find the instructions for this version [here](#).

The System Roles for SAP support multiple hardware architectures for the managed nodes including `x86_64` for Intel compatible nodes, `ppc64le` for IBM Power nodes, and `s390x` for IBM Z Systems.

Important: The System Roles for SAP are designed to be used right after the initial installation of a managed node. Do not run these roles against a SAP or other production system. The role will enforce a certain configuration on the managed node, which might not be intended. Starting with version 3, the roles support an `Assert` parameter for validating existing systems. See “Assert Parameter” on page 216 for more information.

Before applying the roles on a managed node, verify that the RHEL release on the managed node is supported by the SAP software version that you are planning to install.

Configuring SAP systems

The System Roles for SAP roles are designed to set up your SAP systems based on specific SAP documents. Table 4-5 defines what each of the roles are designed to do.

Table 4-5 Role descriptions and use cases.

System Role	Purpose	Use cases
sap_general_preconfigure	Install software and perform all configuration steps which are required for the installation of SAP NetWeaver or SAP HANA.	SAP NetWeaver and SAP HANA
sap_netweaver_preconfigure	Install additional software and perform additional configuration steps which are required for SAP NetWeaver.	SAP NetWeaver
sap_hana_preconfigure	Install additional software and perform additional configuration steps which are required for SAP HANA.	SAP HANA

To prepare a managed node for running SAP HANA you would run both the sap_general_preconfigure role and the sap_hana_preconfigure role. Likewise, to prepare a node to run SAP NetWeaver you would run sap_general_preconfigure role and the sap_netweaver_preconfigure role. Table 4-6 shows the SAP Notes that are implemented by each of the system roles.

Table 4-6 Action performed and SAP Note implemented

System Role	SAP Note for RHEL 7	SAP Note for RHEL 8
sap_general_preconfigure	SAP Note 2002167 SAP Note 1391070 SAP Note 0941735 (TMPFS only)	SAP Note 2772999
sap_netweaver_preconfigure	SAP Note 2526952 (tuned profiles only)	SAP Note 2526952 (tuned profiles only)
sap_hana_preconfigure	Install required packages as per documents SAP HANA 2.0 running on RHEL 7.x and SAP HANA SPS 12 running on RHEL 7.x which are attached to SAP Note 2009879 ppc64le only: Install additional required packages as per https://www14.software.ibm.com/support/cus_tomercare/sas/f/1opdiags/home.html Perform configuration steps as per documents SAP HANA 2.0 running on RHEL 7.x and SAP HANA SPS 12 running on RHEL 7.x which are attached to SAP Note 2009879 ppc64le only: SAP Note 2055470 SAP Note 2292690 SAP Note 2382421	Install required packages for SAP HANA as mentioned in SAP Note 2772999 ppc64le only: Install additional required packages as per https://www14.software.ibm.com/support/cus_tomercare/sas/f/1opdiags/home.html ppc64le only: SAP Note 2055470 SAP Note 2777782 SAP Note 2382421

Assert Parameter

Starting with version 3 of the package rhel-system-roles-sap supports running the roles in assert mode. In assert mode managed nodes are not modified, instead they report the compliance of a node with the applicable SAP notes.

When running playbooks that use the assert mode on previous versions of the roles the assert parameters are ignored which can modify the managed node instead of checking them. Ensure that version 3 of the package is used. In addition check that the playbooks you are using are calling the roles from the correct location which is `/usr/share/ansible/roles` by default.

Preparing a system for SAP HANA installation

To prepare a system, *hana-p11* for SAP HANA installation you can follow these steps:

1. Verify that there is no production software running on the managed node. Generally the RHEL system roles for SAP are run right after RHEL installation so there would be no production software.
2. Validate that you can connect to the managed node using ssh without password.

```
# ssh hana-p11 uname -a
```

3. Create a yml file named `sap-hana.yml` with the content shown in Example 4-21.

Example 4-21 contents of sap-hana.yml

```
- hosts: all
- vars:
  sap_preconfigure_reboot_ok: yes
  sap_hana_preconfigure_enable_sap_hana_repos: yes
  sap_hana_preconfigure_set_minor_release: yes
  sap_hana_preconfigure_modify_grub_cmdline_linux: yes
  sap_hana_preconfigure_reboot_ok: yes
roles:
- sap_general_preconfigure
- sap_hana_preconfigure
```

4. Run the following command:

```
# ansible-playbook -l hana-p11 sap-hana.yml
```

4.7.2 Using the SAP LinuxLab Automation

There are multiple collections in the SAP LinuxLab open-source initiative. These collections can be combined to automate the full lifecycle of your SAP landscapes.

Most of the initial configuration and provisioning activity (Day 0 activities) are done with the Terraform modules and templates which support a wide variety of infrastructures, both on-premises and in the cloud. These options include support for on-premises Power servers and IBM PowerVS cloud instances.

community.sap_install Ansible collection

Day 1 automated SAP installation activity is provided by the `community.sap_install` collection. This Ansible Collection executes various SAP Software installations and configuration tasks for running SAP software on Linux operating systems. It includes handlers for SAP HANA database lifecycle manager (HDBLCM) and SAP Software Provisioning Manager (SWPM) which allows programmatic deployment of any SAP solution scenarios. This can be combined with other Ansible Collections to provide end-to-end automation, from download of SAP software installation media to full configuration of SAP NetWeaver application servers and full HA support utilizing built in SAP HANA system replication technologies.

This Ansible Collection executes various SAP Software installations for different SAP solution scenarios, including:

- ▶ SAP HANA installations via SAP HANA database lifecycle manager (HDBLCM)
 - Install SAP HANA database server, with any SAP HANA Component (e.g. Live Cache Apps, Application Function Library etc.)
 - Configure Firewall rules and Hosts file for SAP HANA database server instance/s
 - Apply license to SAP HANA
 - Configure storage layout for SAP HANA mount points (i.e. /hana/data, /hana/log, /hana/shared)
 - Install SAP Host Agent
 - Install Linux Pacemaker, configure Pacemaker Fencing Agents and Pacemaker Resource Agents
 - Install SAP HANA System Replication
 - Set HA/DR for SAP HANA System Replication
- ▶ Every SAP Software installation via SAP Software Provisioning Manager (SWPM)
 - Run software install tasks using easy Ansible Variable to generate SWPM Unattended installations (sap_swpm Ansible Role default mode).
 - Optional use of template definitions for repeated installations (sap_swpm Ansible Role default templates mode).
 - Run software install tasks with Ansible Variables one-to-one matched to SWPM Unattended ini file parameters to generate bespoke SWPM Unattended installations (sap_swpm Ansible Role advanced mode).
 - Optional use of template definitions for repeated installations (sap_swpm Ansible Role advanced templates mode).
 - Run previously-defined installations with an existing SWPM Unattended inifile.params (sap_swpm Ansible Role inifile_reuse mode)
 - Install Linux Pacemaker, configure Pacemaker Fencing Agents and Pacemaker Resource Agents
 - Set HA/DR with distributed SAP System installations (i.e. ERS)

The following SAP Software solutions have been extensively tested:

- ▶ SAP HANA
 - Scale-Up
 - Scale-Out
 - High Availability
- ▶ SAP NetWeaver AS (ABAP or JAVA) and additional add ons (e.g. GRC, ADS)
- ▶ SAP S/4HANA AnyPremise (1809, 1909, 2020, 2021, 2022)
 - Sandbox (One Host) installation
 - Standard (Dual Host) installation
 - Distributed installation
 - High Availability installation
 - System Copy (Homogeneous with SAP HANA Backup / Recovery) installation
 - Maintenance Planner installation
 - System Rename
- ▶ SAP BW/4HANA

- ▶ SAP Business Suite on HANA (SoH, i.e. SAP ECC on HANA)
- ▶ SAP Business Suite (i.e. SAP ECC with SAP AnyDB - SAP ASE, SAP MaxDB, IBM Db2, Oracle DB)
- ▶ SAP Solution Manager 7.2
- ▶ SAP Web Dispatcher

The collection is designed for Linux operating systems. It has not be tested or adapted for SAP NetWeaver Application Server instances on IBM AIX or Windows Server. It supports Red Hat Enterprise Linux 7 and above and SLES 15 SP3 and above.

Restriction: The collection does not support SLES prior to version 15 SP3 because:

- firewalld, which was added in SLES15 SP3, is used within the Ansible Collection.
- SELinux is used within the Ansible Collection. Full support for SELinux was provided as of SLES 15 SP3.

This collection provides the Ansible roles described in Table 4-7. There are no custom modules.

Table 4-7 Ansible roles provided by the installation collection

Name	Summary
sap_anydb_install_oracle	install Oracle DB 19.x for SAP
sap_general_preconfigure	configure general OS settings for SAP software
sap_ha_install_hana_hsr	install SAP HANA System Replication
sap_ha_pacemaker_cluster	install and configure pacemaker and SAP resources
sap_hana_install	install SAP HANA via HDBLCM
sap_hana_preconfigure	configure settings for SAP HANA database server
sap_hostagent	install SAP Host Agent
sap_hypervisor_node_preconfigure	configure a hypervisor running VMs for SAP HANA
sap_install_media_detect	detect and extract SAP Software installation media
sap_netweaver_preconfigure	configure settings for SAP NetWeaver application server
sap_storage_setup	configure storage for SAP HANA, with LVM partitions and XFS filesystem
sap_swpm	install SAP Software via SWPM
sap_vm_preconfigure	configure settings for a guest (VM) running on RHV/KVM for SAP HANA

Important: In general the “preconfigure” and “prepare” roles are prerequisites for the corresponding installation roles. The logic has been separated to support a flexible execution of the different steps.

community.sap_operations Ansible collection

This collection is designed to assist in operational activity (Day 2) in your SAP landscape. Some of the use cases that can be automated with Ansible for Day 2 operations are:

- ▶ SAP instance system copies.
- ▶ Spin up/delete new application servers on demand (e.g. for hyperscalers).
- ▶ Instance refreshes.
- ▶ Kernel parameter changes.
- ▶ SAP kernel upgrade.
- ▶ DB operations.
- ▶ DB and OS patching.
- ▶ Resource addition (CPU, memory, disk).
- ▶ Cluster management.
- ▶ DB backup/restore.
- ▶ Stop/start of SAP instances.
- ▶ Shutting down of sandbox/preproduction systems to cold storage and pulling them out of storage when needed.
- ▶ Smart management and proactive issue resolution for SAP servers.
- ▶ Near-zero downtime maintenance for SAP servers.

Day 2 operational automation is provided by the community.sap_operations collection. This Ansible Collection executes various SAP Systems operational tasks, which can be used individually during daily operations or can be combined for more complex automation of system maintenance activities. The SAP Systems operational tasks include:

- ▶ OS configuration Post-install of SAP Software
 - Create Ansible user for managing systems
 - Update /etc/hosts file
 - Update SSH authorized known hosts file
 - Update fapolicy entries based on SAP System instance numbers
 - Update firewall port entries based on SAP System instance numbers
 - License registration and refresh for RHEL subscription manager
- ▶ SAP administration tasks
 - Start/Stop of SAP HANA and SAP NetWeaver (in any configuration)
 - Update SAP profile files
 - Execute SAP RFCs

The collection consists of several Ansible roles which combine several Ansible modules into a workflow. These roles, which are shown in Table 4-8, can be used within a playbook for specific tasks.

Table 4-8 Ansible Roles

Name	Summary
os_ansible_user	creates Ansible user ansadm with ssh key
os_etchosts	updates /etc/hosts
os_knownhosts	updates known hosts file /.ssh/known_hosts
sap_control	starting and stopping SAP systems
sap_fapolicy	update service fapolicyd for generic / sap nw / sap hana related uids
sap_firewall	update service firewalld for generic / sap nw / sap hana related ports
sap_profile_update	update default and instance profiles

Name	Summary
sap_rfc	executes SAP RFCs
sap_rhsm	Red Hat subscription manager registration

In addition to the Roles, there are additional Ansible Modules provided by the collection. These modules, shown in Table 4-9, can be called directly within a playbook.

Table 4-9 Ansible Modules

Name	Summary
sap_operations.sap_facts	gather SAP facts in a host (e.g. SAP System IDs and SAP System Instance Numbers of either SAP HANA database server or SAP NetWeaver application server)
sap_operations.sap_monitor_hana_status	check status of running SAP HANA database server
sap_operations.sap_monitor_nw_status	check status of running SAP NetWeaver application server
sap_operations.sap_monitor_nw_perf	check host performance metrics from SAP NetWeaver Primary Application Server (PAS) instance
sap_operations.sap_monitor_nw_response	check system response time metrics from SAP NetWeaver Primary Application Server (PAS) instance

Additional software download automation is provided by the community.sap_launchpad collection.

Example Scenarios

This section provides some example scenarios using the functions provided by the community.sap_operations Ansible collection.

► sap_control

This Ansible Role executes basic SAP administration tasks on Linux operating systems, including:

- Start/Stop/Restart of SAP HANA Database Server
- Start/Stop/Restart of SAP NetWeaver Application Server
- Multiple Automatic discovery and Start/Stop/Restart of SAP HANA Database Server or SAP NetWeaver Application Server

The specific control function is defined using the sap_control_function parameter which can be any of the following:

- restart_all_sap
- restart_all_nw
- restart_all_hana
- restart_sap_nw
- restart_sap_hana
- stop_all_sap
- start_all_sap
- stop_all_nw
- start_all_nw

- stop_all_hana
- start_all_hana
- stop_sap_nw
- start_sap_nw
- stop_sap_hana
- start_sap_hana

Executions specifying all will automatically detect any System IDs and corresponding Instance Numbers. To specify a specific SAP system you would provide the SAP system SID as a parameter.

To restart all SAP systems you would input:

```
sap_control_function: "restart_all_sap"
```

To stop a specific SAP HANA database you would input:

```
sap_control_function: "stop_sap_hana"
sap_sid: "HDB"
```

► sap_hana_sr_takeover

This role can be used to ensure, control and change SAP HANA System Replication. The role assumes that the SAP HANA System Replication was configured using the community.sap_install.sap_ha_install_hana_hsr role.

The variables shown in Table 4-10 are mandatory for running this role unless a default value is specified.

Table 4-10 Required variables

Variable Name	Description
sap_hana_sr_takeover_primary	Server to become the primary server
sap_hana_sr_takeover_secondary	Server to register as secondary. The role can be run twice if more than one secondary is needed by looping over this variable
sap_hana_sr_takeover_sitename	Name of the site being registered as secondary i
sap_hana_sr_takeover_rep_mode	HANA replication mode (defaults to sync if not set)
sap_hana_sr_takeover_hsr_oper_mode	HANA replication operation mode (defaults to logreplay)
sap_hana_sid	HANA SID
sap_hana_instance_number	HANA instance number

The playbook shown in Example 4-22 shows how to implement this role. The assumption is that there are two systems set up for SAP HSR, *hana1* and *hana2*, with SID *RHE* and instance *00*. The playbook ensures that *hana1* is the primary and *hana2* is the secondary. The role will do nothing if *hana1* is already the primary and *hana2* the secondary. The role will fail if *hana1* is not configured for system replication and is not in sync.

Example 4-22 Playbook to set primary in HANA HSR setup

```
---
- name: Ensure hana1 is primary
  hosts: hanas
  become: true
  tasks:
    - name: Switch to hana1
      ansible.builtin.include_role:
        name: community.sap_operations.sap_hana_sr_takeover
```

```
vars:
  sap_hana_sr_takeover_primary: hana2
  sap_hana_sr_takeover_secondary: hana1
  sap_hana_sr_takeover_sitename: DC01
  sap_hana_sid: "RHE"
  sap_hana_instance_number: "00"
```

Additional documentation and examples are available as part of the collection documentation at https://github.com/sap-linuxlab/community.sap_operations. In the roles directory in the GitHub location, there is a subdirectory for each role which includes a readme file providing the specifics of how that role operates and its requirements.

community.sap_launchpad Ansible collection

This Ansible Collection executes basic SAP.com Support operations tasks to assist in downloading SAP software or downloading files from the SAP Maintenance Planner. Maintenance planner is a solution hosted by SAP that helps plan and maintain systems in an SAP landscape and can be used to plan complex activities like installing a new system or updating existing systems.

The operations supported include:

- ▶ Software Center Catalog
 - Search and Download of SAP software center catalog files
 - Search and Extraction of SAP software center catalog information
- ▶ Maintenance Planner
 - Lookup and download files from an existing 'New Implementation' MP Transaction and Stack, using SAP software center's download basket

The collection provides the modules shown in Table 4-11.

Table 4-11 Ansible modules from community.sap_launchpad

Name	Functions
sap_launchpad.software_center_download	search for files and download
sap_launchpad.maintenance_planner_files	maintenance planner files retrieval
sap_launchpad.maintenance_planner_stack_xml_download	maintenance planner stack XML download

Note: SAP software installation media must be obtained from SAP directly, and requires valid license agreements with SAP in order to access these files.

Credentials - SAP User ID

An SAP Company Number (SCN) contains one or more Installation Numbers, providing licenses for specified SAP Software. When an SAP User ID is created within the SAP Customer Number (SCN), the administrator must provide SAP Download authorizations for the SAP User ID.

When an SAP User ID is enabled as part of an SAP Universal ID, then the sap_launchpad Ansible collection must use:

- The SAP User ID
- The password for login with the SAP Universal ID

If a SAP Universal ID is used then the recommendation is to check and reset the SAP User ID 'Account Password' in the SAP Universal ID Account Manager, which will help to avoid any potential conflicts. Example 4-23 on page 224 provides an example playbook to download

specific SAP software using the `sap_launchpad.software_center-download` module. Note that in this playbook, the user is prompted to enter the SAP user ID and password. The playbook could be modified to use variables to enter these values.

Example 4-23 Sample playbook to download software from the SAP software center

```

---
- hosts: all

collections:
  - community.sap_launchpad

pre_tasks:
  - name: Install Python package manager pip3 to system Python
    yum:
      name: python3-pip
      state: present
  - name: Install Python dependencies for Ansible Modules to system Python
    pip:
      name:
        - urllib3
        - requests
        - beautifulsoup4
        - lxml

# Prompt for Ansible Variables
vars_prompt:
  - name: suser_id
    prompt: Please enter S-User
    private: no
  - name: suser_password
    prompt: Please enter Password
    private: yes

# Define Ansible Variables
vars:
  ansible_python_interpreter: python3
  softwarecenter_search_list:
    - 'SAPCAR_1324-80000936.EXE'
    - 'HCMT_057_0-80003261.SAR'

# Use task block to call Ansible Module
tasks:
  - name: Execute Ansible Module to download SAP software
    community.sap_launchpad.software_center_download:
      suser_id: "{{ suser_id }}"
      suser_password: "{{ suser_password }}"
      softwarecenter_search_query: "{{ item }}"
      dest: "/tmp/"
      loop: "{{ softwarecenter_search_list }}"
      loop_control:
        label: "{{ item }} : {{ download_task.msg }}"
      register: download_task
      retries: 1
      until: download_task is not failed

```

In Example 4-24, we show an example playbook that downloads a list of files which are defined using the maintenance planner. The playbook prompts for your SAP user credentials and a specific maintenance planner transaction name which has been previously created.

Example 4-24 Playbook to download files defined in a maintenance planner transaction

```

---
- hosts: all

```

```
collections:
  - community.sap_launchpad

# pre_tasks:

# Prompt for Ansible Variables
vars_prompt:
  - name: suser_id
    prompt: Please enter S-User
    private: no
  - name: suser_password
    prompt: Please enter Password
    private: yes
  - name: mp_transaction_name
    prompt: Please enter MP transaction name
    private: no

# Define Ansible Variables
vars:
  ansible_python_interpreter: python3

# Use task block to call Ansible Module
tasks:
  - name: Execute Ansible Module 'maintenance_planner_files' to get files from MP
    community.sap_launchpad.maintenance_planner_files:
      suser_id: "{{ suser_id }}"
      suser_password: "{{ suser_password }}"
      transaction_name: "{{ mp_transaction_name }}"
      register: sap_maintenance_planner_basket_register

  # - debug:
  #   msg:
  #     - "{{ sap_maintenance_planner_basket_register.download_basket }}"

  - name: Execute Ansible Module 'software_center_download' to download files
    community.sap_launchpad.software_center_download:
      suser_id: "{{ suser_id }}"
      suser_password: "{{ suser_password }}"
      download_link: "{{ item.DirectLink }}"
      download_filename: "{{ item.Filename }}"
      dest: "/tmp/test"
    loop: "{{ sap_maintenance_planner_basket_register.download_basket }}"
    loop_control:
      label: "{{ item }} : {{ download_task.msg }}"
    register: download_task
    retries: 1
    until: download_task is not failed
```



Infrastructure as Code Using Ansible

Infrastructure as Code (IaC) is the ability to automate the management of infrastructure resources using code, in our case Ansible. Traditionally we see an Ansible client as a virtual machine, or a storage controller, but with IaC the Ansible client tends to be a service capable of managing the end-to-end infrastructure resources, including virtual machines, networks, storage, zoning, and other resources required. Within IBM Power environments these services include IBM PowerVC, IBM Cloud PowerVS and IBM PowerVM – which provides VIO Server management and the HMC.

Infrastructure as Code provides us the ability to not only deploy or destroy new workloads, but also to resize, re balance and migrate those workloads to different infrastructure. In this chapter we will introduce the concept of Infrastructure as Code and describe the capabilities that Ansible provides to deliver IaC on IBM Power.

The following topics are covered in this chapter:

- ▶ IBM Power Virtualization Center
- ▶ IBM Power Virtual Server (PowerVS)

5.1 IBM Power Virtualization Center

IBM Power Virtualization Center (PowerVC) provides simplified management of IBM AIX, IBM i and Linux virtual machines (VMs) running on IBM Power. It is built on OpenStack to provide private cloud capabilities across your IBM Power environment. IBM PowerVC capabilities include being able to create and destroy VMs, networks, network interfaces, storage volumes and images. It also has the ability to perform tasks against the VMs such as stop, start, resize, migrate, clone, create and restore snapshots, and attach storage volumes.

Advantages of PowerVC

PowerVC offers a range of benefits tailored to IBM Power environments:

- ▶ **Expandability:** Attach volumes or additional networks to VMs.
- ▶ **Flexibility:** Import and export existing systems and volumes between on-premises and off-premises locations.
- ▶ **Efficiency:** Take snapshots of VMs and clone them for quick replication.
- ▶ **Seamless Migration:** Migrate running VMs using Live Partition Mobility (LPM).
- ▶ **Continuity:** Restart VMs remotely in the event of a server failure.
- ▶ **Simplified Management:** Streamline Power Systems virtualization administration.
- ▶ **Agility:** Adapt swiftly to changing business requirements.
- ▶ **Dynamic Resource Management:** Create, resize, and adjust CPU and memory resources for VMs.

When we deploy a new VM using IBM PowerVC it performs all of the required tasks. These include:

- ▶ Creating the VM profile on the HMC, including all network and storage interfaces
- ▶ Creating the appropriate SAN zoning
- ▶ Creating the VM on the storage controller
- ▶ Creating the root and non-root storage volumes
- ▶ Updating the VIO Server to map the VM to its new volumes
- ▶ Booting the new VM

Likewise when we delete a VM, all the resources created by IBM PowerVC, are removed cleanly.

With IBM PowerVC there two options to choose from to work with Ansible, these are:

- ▶ Using the OpenStack modules
- ▶ making RESTful API calls using the URI module

In this section we will cover both methods.

5.1.1 Using the OpenStack Cloud modules

In this section we will be covering the following IaC options using the OpenStack Cloud modules and IBM PowerVC:

- ▶ Authentication
- ▶ Creating a new VM
- ▶ Destroying an existing VM
- ▶ Showing resource information
- ▶ Stopping/Starting a VM
- ▶ Creating a new storage volume
- ▶ Attaching a storage volume to an existing VM

As IBM PowerVC is built on OpenStack we are able to use a number of the cloud modules provided by the OpenStack community.

These are available on [Ansible Galaxy](#).

You can download the OpenStack Cloud collection either by using **ansible-galaxy** from the command line, or via a **requirements.yml** file. Example 5-1 shows using **ansible-galaxy** to download the collection.

Example 5-1 Downloading the OpenStack Cloud collection

```
ansible-galaxy collection install openstack.cloud
Process install dependency map
Starting collection install process
Installing 'openstack.cloud:2.1.0' to
'/root/.ansible/collections/ansible_collections/openstack/cloud'
```

Example 5-2 shows the **requirements.yml** that you can use to download the collection.

Example 5-2 Example requirements.yml file to download the OpenStack Cloud collection

```
collections:
  - name: openstack.cloud
    source: https://galaxy.ansible.com
```

In order for Ansible to run the OpenStack Cloud modules, you must first install the OpenStack SDK on your Ansible controller. This is described in the 'Read Me' section of the collection page in galaxy. The command shown in Example 5-3 will install the SDK.

Example 5-3 Downloading the OpenStack Cloud collection

```
$ pip3.9 install openstacksdk
```

You can verify that the modules have been installed correctly by using the **ansible-doc** command. An example of viewing the documentation for the OpenStack Cloud image info module is shown in Example 5-4.

Example 5-4 Viewing the openstack.cloud.server_info documentation

```
$ ansible-doc openstack.cloud.image_info
..
REQUIREMENTS: python >= 3.6, openstacksdk >= 1.0.0
AUTHOR: OpenStack Ansible SIG
EXAMPLES:
- name: Gather previously created image named image1
  openstack.cloud.image_info:
    cloud: devstack-admin
    image: image1

- name: List all images
  openstack.cloud.image_info:
```

Table 5-1 on page 230 shows some of the OpenStack Cloud modules relevant to IBM PowerVC.

Table 5-1 OpenStack Cloud Modules

Modules name	Function
openstack.cloud.auth	Retrieve auth token from PowerVC Cloud
openstack.cloud.compute_flavor	Manage PowerVC compute flavors
openstack.cloud.compute_flavor_info	Fetch compute flavors information from PowerVC Cloud
openstack.cloud.image	Manage PowerVC images
openstack.cloud.image_info	Fetch image information from PowerVC Cloud
openstack.cloud.keypair	Manage PowerVC keypairs
openstack.cloud.keypair_info	Fetch keypair information from PowerVC Cloud
openstack.cloud.project	Manage PowerVC projects
openstack.cloud.project_info	Fetch project information from PowerVC Cloud
openstack.cloud.server	Create/delete VMs within PowerVC Cloud
openstack.cloud.server_action	Perform actions on PowerVC VMs
openstack.cloud.server_info	Fetch VM information from PowerVC Cloud
openstack.cloud.server_volume	Attach/detach volumes from PowerVC VMs
openstack.cloud.volume	Manage PowerVC storage volumes
openstack.cloud.volume_info	Fetch storage volume information from PowerVC Cloud
openstack.cloud.volume_snapshot	Manage PowerVC snapshots
openstack.cloud.volume_snapshot_info	Fetch snaphost information from PowerVC Cloud

Authenticating with IBM PowerVC

Before Ansible can use the OpenStack cloud modules it needs to authenticate with the IBM PowerVC server.

To authenticate from the command line create a 'clouds.yml' file which contains the information about the cloud environments that Ansible needs to connect to. In this case it would be our IBM PowerVC server. The OpenStack modules will look for the clouds.yml file in the following directories:

- current directory
- ~/.config/openstack
- /etc/openstack

It will use the first one it finds. The contents of an example clouds.yml file are shown in Example 5-5.

Example 5-5 Example clouds.yml file

```
$ cat clouds.yml
powervc_cloud:
  auth:
    auth_url: https://x.x.x.x:5000/v3/
    project_name: ibm-default
    project_domain_name: Default
    user_domain_name: Default
    username: <powervc_userid>
    password: <powervc_userid_password>
```

```
region_name: RegionOne
cacert: "./powervc.crt"
```

The first line is the name of your cloud and is for reference only, it does not have to match the real name. The cloud name allows us to define authentication methods to multiple OpenStack clouds and refer to them individually within our playbooks.

The 'auth_url' is the IP address of your IBM PowerVC server, and the remaining auth settings are specific to your PowerVC environment such as project, userid, password etc.

You also need to have a copy of the CA cert file from the PowerVC server that you reference in the 'cacert' line. This can be found on your PowerVC server, the default location is *'etc/pki/tls/certs/powervc.crt'*.

Confirm authentication to IBM PowerVC

Now that we have setup authentication, we can confirm that Ansible can use the OpenStack cloud modules to retrieve information from IBM PowerVC. A simple way to do this is create a playbook to show the images contained on IBM PowerVC. This is shown in Example 5-6.

Example 5-6 Example playbook to list PowerVC images using the openstack.cloud.image_info module

```
$ cat PowerVC_list_images.yml
---
- name: List available PowerVC Images
  hosts: localhost
  gather_facts: false
  tasks:
    - name: Retrieve list of all AIX images
      openstack.cloud.image_info:
        cloud: powervc_cloud
        register: image_results

    - name: Show name, ID, OS distribution and status of images
      debug:
        msg: "{{ image_results | json_query('images[*].
              {name: name, id: id, os_distro: os_distro, status: status}') }}"
```

In Example 5-7 we call a task using the *'openstack.cloud.image_info'* modules, pointing at the cloud *'powervc_cloud'*, This cloud name has to match the entry in our clouds.yml authentication file defined earlier. We then register the results and output them in the second task. We can now run the playbook to list our IBM PowerVC images as shown in Example 5-7.

Example 5-7 Example playbook to list PowerVC images using the openstack.cloud.image_info module

```
$ ansible-playbook PowerVC_list_images.yml
PLAY [List available PowerVC Images]
*****
TASK [Retrieve list of all AIX images]
*****
ok: [localhost]
TASK [Show name, ID, OS distribution and status of images]
*****
ok: [localhost] => {
  "msg": [
    {
      "id": "f62a76dd-4742-445f-aa5c-f3f447dd778e",
      "name": "RHCOS-4.12.17",
```

```

        "os_distro": "coreos",
        "status": "active"
    },
    {
        "id": "0930d057-dc7e-415f-97cd-1fe36ecdcbd",
        "name": "RHEL v9.1",
        "os_distro": "rhel",
        "status": "active"
    },
    {
        "id": "d51d8cfd-c83b-4ec6-9464-8d4215259546",
        "name": "AIX 7.3",
        "os_distro": "aix",
        "status": "active"
    },
    {
        "id": "c64ff508-3a81-4679-a9e4-29acc3f96430",
        "name": "IBM i v7.3",
        "os_distro": "ibmi",
        "status": "active"
    }
]
}

```

```
PLAY RECAP *****
localhost : ok=2 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
```

Note: You may have to install the `community.general` collection to parse the data using `'json_query'`. This is shown in Example 5-8.

Example 5-8 Downloading the Community General collection sing ansible-galaxy

```

$ ansible-galaxy collection install community.general
Starting galaxy collection install process
Process install dependency map
Starting collection install process
Downloading https://galaxy.ansible.com/download/community-general-7.2.1.tar.gz to
/root/.ansible/tmp/ansible-local-3467397n4zdigf9/tmpc_m7f9qt/community-general-7.2.1-i83jeq
8b
Installing 'community.general:7.2.1' to
'/root/.ansible/collections/ansible_collections/community/general'
community.general:7.2.1 was installed successfully

```

Creating VMs using the `openstack.cloud.server` module

We can use the `openstack.cloud.server` module to create an AIX, IBM i or Linux VM as shown in Example 5-9.

Example 5-9 Task to create a new VM using PowerVC

```

- name: Creating VM {{ VM_Name }} using PowerVC
  openstack.cloud.server:
    cloud: "{{ PowerVC_Cloud_Name }}"
    state: present
    name:   "{{ VM_Name }}"
    image: "{{ Image_ID_or_Name }}"
    flavor: "{{ Flavor_ID_or_Name }}"
    network: "{{ Network_Name }}"
    key_name: "{{ SSH-Key_Name }}"
  register: vm_create_information

```

Note that in Example 5-9 on page 232 we passed the `openstack.cloud.server` module a few key variables to allow it to build the VM.

- ▶ `cloud` - The name of the PowerVC cloud defined in `clouds.yml`
- ▶ `state` - `present` (if the VM does not exist, create it)
- ▶ `name` - name of the new VM (in this example we've used 'aix-vm-1')
- ▶ `image` - name or ID of the PowerVC image to use (obtained from PowerVC)
- ▶ `flavor` - name or IR of the PowerVC compute flavor to use (obtained from PowerVC)
- ▶ `network` - name of the PowerVC network to use (obtained from PowerVC)
- ▶ `key_name` - name of the SSH key pair to inject into the new VM (obtained from PowerVC)

When we run this task, Ansible will use the `openstack.cloud.server` module to connect to the IBM PowerVC cloud and create the VM using the name provided. The results for running our playbook are shown in Example 5-10.

Example 5-10 Output from create VM using OpenStack modules on IBM PowerVC

```
PLAY [Connect to PowerVC/Openstack and build VM]
*****

TASK [Creating VM aix-vm-1 using PowerVC]
*****
changed: [localhost]
PLAY RECAP *****
localhost : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0
ignored=0
```

We can also see the new VM being created on the IBM PowerVC UI, as shown in Figure 5-1.

Note: In Example 5-10 we have allowed PowerVC to assign an IP address from its IP pool.

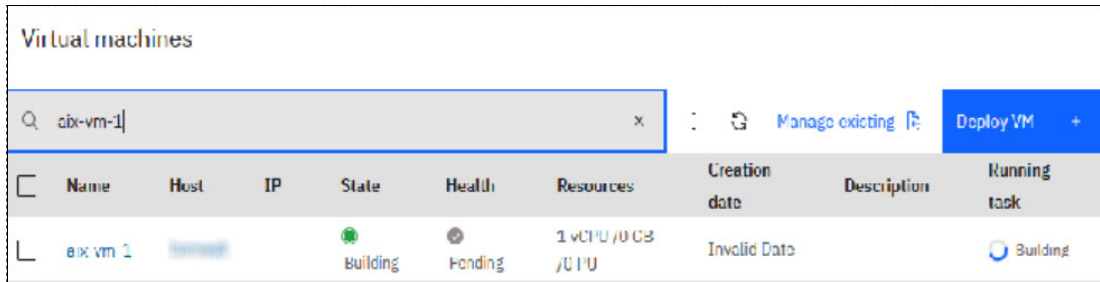


Figure 5-1 PowerVC VM build using OpenStack modules

If any of the variables passed are incorrect, the module will fail without creating the VM. For example if you passed the module an incorrect image name, it would fail with a message similar to that shown in Example 5-11.

Example 5-11 Error showing incorrect PowerVC image name

```
TASK [Create a new VM instance using PowerVC]
*****
fatal: [localhost]: FAILED! => {"changed": false, "msg": "Could not find image AIX 7.2 TL2 SP6"}
```

Destroy a VM using the openstack.cloud.server module

We can also use the openstack.cloud.server module to destroy an AIX, IBM i or Linux VM as shown Example 5-12.

Example 5-12 Task to destroy an existing VM using PowerVC

```
- name: "Destroying VM {{ VM_Name }} using PowerVC"
  openstack.cloud.server:
    cloud: "{{ PowerVC_Cloud_Name }}"
    state: absent
    name:   "{{ VM_Name }}"
  register: vm_destroy_information
```

In Example 5-12 we only had to pass the openstack.cloud.server module three variables to destroy the VM:

- ▶ cloud - The name of the PowerVC cloud defined in clouds.yml
- ▶ state - absent (if the VM exists, remove it)
- ▶ name - name of the existing VM to destroy (in this example we've used 'aix-vm-1')

As the VM is managed by IBM PowerVC, by default when it is destroyed all of its resources including the storage volumes, the SAN zones and its IP allocations are also removed.

When we use the openstack.cloud.server module to destroy an existing VM we simply get the message that the status was changed, as shown in Example 5-13.

Example 5-13 Message showing existing VM destroyed via PowerVC

```
TASK [Destroying VM aix-vm-1 using PowerVC]
*****
changed: [localhost]

TASK [Show VM destroy output]
*****
ok: [localhost] => {
  "vm_destroy_information": {
    "changed": true,
    "failed": false
  }
}
PLAY RECAP *****
localhost : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0
ignored=0
```

We can also see the new VM being destroyed on the IBM PowerVC UI, as shown in Figure 5-2.

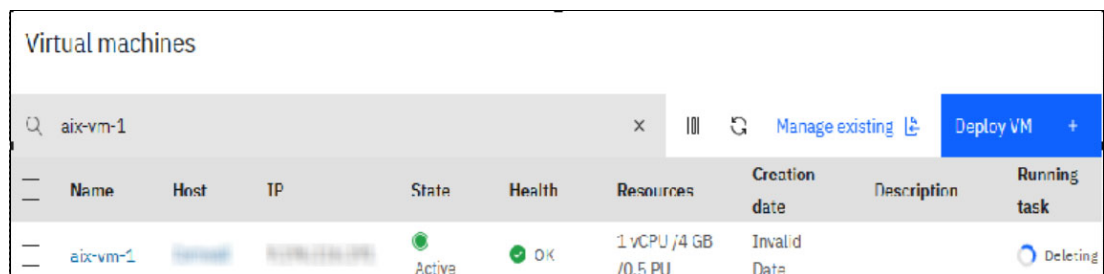


Figure 5-2 PowerVC UI showing VM destroy using OpenStack modules

If we try and destroy a VM that does not exist, the `openstack.cloud.server` module does not by default return an error, as we stated the VM should be of `'state: absent'`. It simply tells us the status has not changed (`false`), as shown in Example 5-14.

Example 5-14 Message showing attempt to destroy a VM that does not exist via PowerVC

```
TASK [Destroying VM aix-vm-1 using PowerVC]
*****
ok: [localhost]

TASK [Show VM destroy output]
*****
ok: [localhost] => {
  "vm_information": {
    "changed": false,
    "failed": false
  }
}
```

Retrieve IBM PowerVC resource information using the `openstack.cloud` modules

Along with being able to create and destroy PowerVC resources such as VMs, storage volumes, projects, networks etc, we can also use the `openstack.cloud` module to retrieve data from our PowerVC cloud. We can see some of the common 'information' modules in Table 5-1 on page 230, including `openstack.cloud.server_info`, `openstack.cloud.volume_info`, `openstack.cloud.image_info`. Because the `openstack.cloud` modules are retrieving their data from the PowerVC server, and not the actual VM itself, we are able to query the resources infrastructure.

Display all VM information via PowerVC

We can use the `openstack.cloud.server_info` to retrieve infrastructure information about a PowerVC controlled VM. In Example 5-15 we show the module collecting all the information that PowerVC knows about a VM.

Example 5-15 Task to display all VM information

```
- name: Retrieve all information about an existing VM instance using PowerVC
  openstack.cloud.server_info:
    cloud: "{{ PowerVC_Cloud_Name }}"
    name:   "{{ VM_Name }}"
    register: vm_information

- name: Show all VM information collected
  var: vm_information
```

The output collected shows a large amount of information that PowerVC was able to retrieve about the VM and is shown in Example 5-16.

Example 5-16 Output from `openstack.cloud.server_info`

```
Output of information retrieve about a VM using openstack.cloud.server_info
"servers": [
  {
    "access_ipv4": "x.x.x.x",
    "access_ipv6": "",
    "addresses": {
      "VLAN_888-NET_116": [
        {
```

```

        "OS-EXT-IPS-MAC:mac_addr": "fa:5f:75:e3:yy:xx",
        "OS-EXT-IPS:type": "fixed",
        "addr": "x.x.x.x",
        "version": 4
    }
]
},
"admin_password": null,
"attached_volumes": [
    {
        "attachment_id": null,
        "bdm_id": null,
        "delete_on_termination": true,
        "device": null,
        "id": "5119dfc1-8fc2-4a70-943d-da6266d71f9b",
        "location": null,
        "name": null,
        "tag": null,
        "volume_id": null
    },
    {
        "attachment_id": null,
        "bdm_id": null,
        "delete_on_termination": false,
        "device": null,
        "id": "25d70a6e-0923-491c-bb87-47b484b11c16",
        "location": null,
        "name": null,
        "tag": null,
        "volume_id": null
    }
],
"availability_zone": "Default Group",
"block_device_mapping": null,
"compute_host": "828422A_XXXXXX",
"config_drive": "",
"created_at": "2023-06-28T09:09:50Z",
"description": "aix-vm-1",
"disk_config": "MANUAL",
"fault": null,
"flavor": {
    "description": null,
    "disk": 0,
    "ephemeral": 0,
    "extra_specs": {
        "powervm:availability_priority": "127",
        "powervm:dedicated_proc": "false",
        "powervm:enable_lpar_metric": "true",
        "powervm:enforce_affinity_check": "false",
        "powervm:max_mem": "4096",
        "powervm:max_proc_units": "0.5",
        "powervm:max_vcpu": "1",
        "powervm:min_mem": "2048",
        "powervm:min_proc_units": "0.1",
        "powervm:min_vcpu": "1",
        "powervm:proc_units": "0.1",
        "powervm:processor_compatibility": "default",
        "powervm:secure_boot": "0",
        "powervm:shared_proc_pool_name": "DefaultPool",
        "powervm:shared_weight": "128",
    }
}

```

```

        "powervm:srr_capability": "true",
        "powervm:uncapped": "true"
    },
    "id": "xtiny",
    "is_disabled": null,
    "is_public": true,
    "location": null,
    "name": "xtiny",
    "original_name": "xtiny",
    "ram": 4096,
    "rxtx_factor": null,
    "swap": 0,
    "vcpus": 1
},
"flavor_id": null,
"has_config_drive": "",
"host_id": "6e82dcb4ed92b0e70c305e2ee1021f0019d3bd88e9dd910b5a81xxxx",
"host_status": "UP",
"hostname": "aix-vm-1",
"hypervisor_hostname": "XXXXX",
"id": "371aa5fe-b5c2-4660-978b-09b323a49f66",
"image": {
    "architecture": null,
    "checksum": null,
    "container_format": null,
    "created_at": null,
    "direct_url": null,
    "disk_format": null,
    "file": null,
    "has_auto_disk_config": null,
    "hash_algo": null,
    "hash_value": null,
    "hw_cpu_cores": null,
    "hw_cpu_policy": null,
    "hw_cpu_sockets": null,
    "hw_cpu_thread_policy": null,
    "hw_cpu_threads": null,
    "hw_disk_bus": null,
    "hw_machine_type": null,
    "hw_qemu_guest_agent": null,
    "hw_rng_model": null,
    "hw_scsi_model": null,
    "hw_serial_port_count": null,
    "hw_video_model": null,
    "hw_video_ram": null,
    "hw_vif_model": null,
    "hw_watchdog_action": null,
    "hypervisor_type": null,
    "id": "71c5ddb5-f4f9-431b-917d-e0c0df581xxx",
    "instance_type_rxtx_factor": null,
    "instance_uuid": null,
    "is_hidden": null,
    "is_hw_boot_menu_enabled": null,
    "is_hw_vif_multiqueue_enabled": null,
    "is_protected": null,
    "kernel_id": null,
    "location": null,
    "locations": null,
    "metadata": null,
    "min_disk": null,

```

```

    "min_ram": null,
    "name": null,
    "needs_config_drive": null,
    "needs_secure_boot": null,
    "os_admin_user": null,
    "os_command_line": null,
    "os_distro": null,
    "os_require_quiesce": null,
    "os_shutdown_timeout": null,
    "os_type": null,
    "os_version": null,
    "owner": null,
    "owner_id": null,
    "properties": {
      "links": [
        {
          "href":
"https://x.x.x.x:8774/6a01a6c6f13c40f79b7ff55xxxx70a371/images/71c5ddb5-f4f9-431b-917d-e0c0
xxx",
          "rel": "bookmark"
        }
      ]
    },
    "ramdisk_id": null,
    "schema": null,
    "size": null,
    "status": null,
    "store": null,
    "tags": [],
    "updated_at": null,
    "url": null,
    "virtual_size": null,
    "visibility": null,
    "vm_mode": null,
    "vmware_adaptertype": null,
    "vmware_ostype": null
  },
  "image_id": null,
  "instance_name": "aix-vm-1-371aa5fe-00000b9e",
  "is_locked": false,
  "kernel_id": "",
  "key_name": "ssh-key",
  "launch_index": 0,
  "launched_at": "2023-06-28T09:13:22.000000",}
],
"max_count": null,
"metadata": {
  "enforce_affinity_check": "false",
  "hostname": "aix-vm-1",
  "move_pin_vm": "false",
  "original_host": "828422A_xxxxx",
},
"min_count": null,
"name": "aix-vm-1",
"networks": null,
"power_state": 1,
"progress": 100,
"project_id": "6a01a6c6f13c40f79b7ff5552170axxx",
"ramdisk_id": "",
"reservation_id": "r-4n2mezi3",

```

```

    "root_device_name": "/dev/sda",
    "scheduler_hints": null,
    "security_groups": null,
    "server_groups": null,
    "status": "ACTIVE",
    "tags": [],
    "task_state": null,
    "terminated_at": null,
    "trusted_image_certificates": null,
    "updated_at": "2023-08-15T13:08:46Z",
    "user_data": null,
    "user_id":
"0688b01e6439ca32d698d20789d52169126fb41fb1a4ddafcebb97d854e836c9",
    "vm_state": "active",
    "volumes": [
      {
        "attachment_id": null,
        "bdm_id": null,
        "delete_on_termination": true,
        "device": null,
        "id": "5119dfc1-8fc2-4a70-943d-da6266d71f9b",
        "location": null,
        "name": null,
        "tag": null,
        "volume_id": null
      },
      {
        "attachment_id": null,
        "bdm_id": null,
        "delete_on_termination": false,
        "device": null,
        "id": "25d70a6e-0923-491c-bb87-47b484b11c16",
        "location": null,
        "name": null,
        "tag": null,
        "volume_id": null
      }
    ]

```

Display only VMs hosted on a specific IBM Power server via PowerVC

Using the output shown in Example 5-16 on page 235 we are able to select which VMs we want to display by filtering on items such as status, network, image or hosted IBM Power server. We are also able to select which values we display in our output. In Example 5-17 we use the `openstack.cloud.server_info` module to retrieve information about VMs on a certain IBM Power server, and display the name, status and memory allocation of those VMs.

Example 5-17 Display name, status and memory of all VMs on a specific IBM Power Server

```

- name: Collect information of all VMs on PowerServer1 via PowerVC
  openstack.cloud.server_info:
    cloud: powervc_cloud
    filters:
      compute_host: "{{ Server_serial_number }}"
    register: vm_on_host_results

- name: Show name, status and memory of VMs on PowerServer1
  debug:
    msg: "{{ vm_on_host_results | json_query('servers[*].
      {name: name, status: vm_state, memory: flavor.ram}') }}"

```

```
TASK [Show name, status and memory of VMs on PowerServer1]
*****
ok: [localhost] => {
  "msg": [
    {
      "memory": 4096,
      "name": "aix-vm-1",
      "status": "active"
    },
    {
      "memory": 4096,
      "name": "ibmi-vm-1",
      "status": "active"
    }
  ]
}
```

Stop/Start a PowerVC VM using the openstack.cloud modules.

Another useful module from the OpenStack Cloud collection, is the 'server_action' module. This allows you to perform a stop or start action against an existing VM via PowerVC.

In Example 5-18 we show an example of using openstack.cloud.server_action.

Example 5-18 Playbook for stopping or starting a PowerVC VM

```
- name:
  openstack.cloud.server_action:
    cloud: powervc_cloud
    name: "{{ VM_Name }}"
    action: <stop/start>
    register: result
```

Create and attach a storage volume using the openstack.cloud modules

Using the openstack.cloud modules we are able to create and attach a new volume in IBM PowerVC. We do this using two of the modules from the collection:

- ▶ openstack.cloud.volume - to create the volume
- ▶ openstack.cloud.server_volume - to attach the volume to an existing VM

The 'openstack.cloud.volume' module documentation can be found [here](#):

In Example 5-19 we show an example of using this module to create a 10GB storage volume.

Example 5-19 Create a 10GB storage volume using the OpenStack Cloud collection

```
- name: Create a new {{ new_disk_size }}GB volume, called {{ new_disk_name }} using storage
template {{ storage_template }}
  openstack.cloud.volume:
    cloud: powervc_cloud
    state: present
    name: "{{ new_disk_name }}"
    size: "{{ new_disk_size }}"
    volume_type: "{{ storage_template }}"
    register: volume_create_information
```

Note: The 'size' of the volume is in GB, and the 'volume_type' refers to the PowerVC storage template to use.

Once we have created the new storage volume we can attach it to an existing VM. We do this using the OpenStack Cloud server volume module. The documentation for that modules can be found [here](#):

In Example 5-20 we show an example of using this module to attach the volume to an existing VM in PowerVC.

Example 5-20 Attach a storage volume to an existing PowerVC VM

```
- name: "Attach storage volume {{ new_disk_name }} to VM {{ VM_Name }}"
  openstack.cloud.server_volume:
    cloud: powervc_cloud
    state: present
    server: "{{ VM_Name }}"
    volume: "{{ new_disk_name }}"
    register: volume_attach_information
```

We can combine both tasks in the same playbook to first create, then attach the new storage to an existing VM, as shown in Example 5-21 and Figure 5-3.

Example 5-21 Output showing create and attach a new storage volume via PowerVC

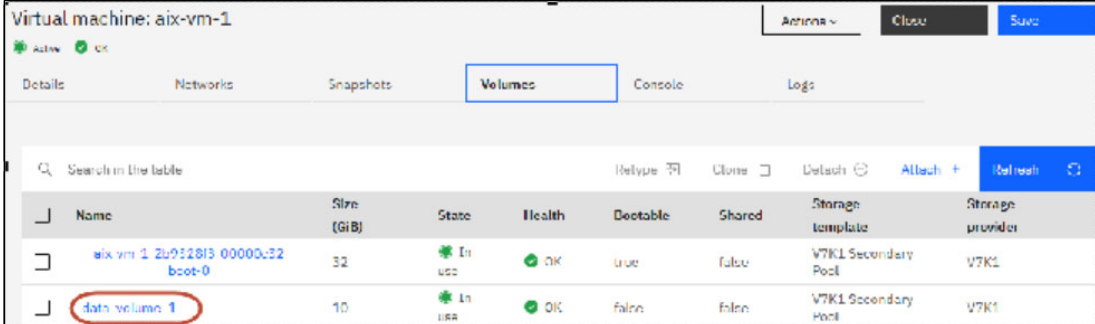
```
PLAY [Connected to PowerVC/Openstack VM, create new disk and attach to VM]
*****

TASK [Create a new 10GB volume, called data_volume_1 using storage template V7K1 Secondary Pool] *****
changed: [localhost]

TASK [Attach storage volume data_volume_1 to VM aix-vm-1]
*****
changed: [localhost]

PLAY RECAP *****
localhost : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0
ignored=0
```

The results can be seen in the PowerVC user interface as shown in Figure 5-3.



Name	Size (GiB)	State	Health	Detachable	Shared	Storage template	Storage provider
aix-vm-1 2b932815 00000c32 boot-0	32	In use	OK	true	false	V7K1 Secondary Pool	V7K1
data volume 1	10	In use	OK	false	false	V7K1 Secondary Pool	V7K1

Figure 5-3 PowerVC UI showing new volume attached to VM

5.1.2 Using the URI modules to interact with PowerVC API services

Another method of automating IBM PowerVC using Ansible is to take advantage of the REST (Representational State Transfer) APIs IBM PowerVC provides. The OpenStack software has industry-standard interfaces that are released under the terms of the Apache License. IBM PowerVC interfaces are a subset of OpenStack northbound APIs.

In this section we will be covering the following IaC options using the URI module utilizing PowerVC API services:

- ▶ Authentication
- ▶ Creating a new VM
- ▶ Destroying an existing VM
- ▶ Showing resource information
- ▶ Resizing an online VM

A number of interfaces were added or extended to enhance the capabilities that are associated with the IBM Power platform REST APIs.

APIs use a common set of methods which we will use to perform operations on IBM PowerVC.

- ▶ POST - Create operation
- ▶ GET - Read operation
- ▶ PUT - Update operation
- ▶ DELETE - Delete operation

There are three types of APIs to integrate Ansible with PowerVC:

- ▶ **Supported OpenStack APIs:** These APIs are a subset of the APIs provided by OpenStack and can be used with PowerVC without any modifications.
- ▶ **Extended OpenStack APIs:** These APIs are a subset of the APIs provided by OpenStack, but their functions are extended by PowerVC.
- ▶ **PowerVC APIs:** These APIs do not exist in OpenStack and are exclusive to PowerVC.

PowerVC APIs are provided by a number of specialized inter-operable services. Each service is accessible on a distinct port number and provides a set of APIs that run specialized functions that are related to that service. The services are shown in Table 5-2.

Table 5-2 PowerVC API Services

Project or service name	Description
OpenStack Projects	
Telemetry (ceilometer)	Billing, benchmarking, scalability, and statistics. Is used for auditing in PowerVC.
Storage (cinder)	Storage and storage volume management.
Image (glance)	Images and image management.
Identity (keystone)	Security, identity, and authentication services.
Networking (neutron)	Networking and network management.
Compute (nova)	Host or compute. Manages the lifecycle and operations of compute resources.
PowerVC services	
Validator	Validates the PowerVC environment.

The OpenStack APIs as shown in Table 5-2 include the ability to read, create, update and delete IBM PowerVC resources including VMs, networks, storage, key pairs, images and projects. The reference documentation can be found at the [OpenStack organization site](#).

The IBM PowerVC APIs (along with references to the OpenStack APIs) are documented in the [PowerVC documentation](#).

Each OpenStack and IBM PowerVC API service uses a unique port. Some of the key API ports are shown in Table 5-3.

Table 5-3 PowerVC API service ports

Service	Function	Port
Keystone	Identify/authentication	5000
Nova	Compute	8774
Neutron	Network	9696
Glance	Images	9292
Cinder	Storage	9000

To access IBM PowerVC APIs via Ansible, we can use the ‘uri’ module, which is part of ansible-core (ansible.builtin.uri).

Authenticating with IBM PowerVC (API)

The first thing we have to do before we can perform any API actions on IBM PowerVC is obtain an authentication token.

To do this we have to perform an API POST to the PowerVC server with the following information:

- ▶ API URL of the PowerVC server (IP or hostname)
- ▶ Keystone authentication port (default 5000)URI
- ▶ PowerVC user name and password
- ▶ Tenant/Project name
- ▶ Domain name (only ‘default’ is supported)

Example 5-22 shows Ansible URI module authenticating with IBM PowerVC, obtaining the information required, setting a fact to store the authorization token, then displaying the token.

Example 5-22 Obtaining the authorization token from PowerVC using URI module

```
- name: Connect to PowerVC and collect auth token
  uri:
    url: https://{ powervc_host }:{ auth_port }/v3/auth/tokens
    method: POST
    body: '{"auth":{
      "scope":{
        "project":{
          "domain":{
            "name":"Default",
            "name":"ibm-default"}},
        "identity":{
          "password":{
            "user":{
              "domain":{
                "name":"Default",
                "password":"{{ PowerVC_password }}",
                "name":"{{ PowerVC_ID }}"}},
            "methods":["password"]}}}'
    body_format: json
    use_proxy: no
```

```

        validate_certs: no
        status_code: 201
        register: auth
- name: Set Auth Token
  set_fact:
    auth_token: "{{ auth.x_subject_token }}"
- name: Display Auth Token
  debug:
    var: auth_token

```

Although we wouldn't normally display the token, we do it in this case to demonstrate that Ansible has been able to authenticate with the PowerVC server. The output is shown in Example 5-23.

Example 5-23 Authorization token output

```

- TASK [Connect to PowerVC and collect auth token]
*****
ok: [localhost]

TASK [Set Auth Token]
*****
ok: [localhost]

TASK [Display Auth Token]
*****
ok: [localhost] => {
  "auth_token":
    "gAAAAABkZ6Y92U11u0uFuXzmv7JdU1-st3SPkf_1wTTQRE2sm8yATw6KRMU9vGHtIJHaT5ZHgk18cHLdzRwoLqQhL
    tByBhKEw96-pKBfMD0PfsWTaJiTsrAmddRaqM18Y4b4ZbmFrESaTI4pzzZH2uHIEby0KhPSm7-Wn5A58gg2RAa0ARY3
    MrgeIHVvPDrMKOD3G1qwHv1-GNPFwZaqkZzKwM9XXXXXXXXXXXXXXXXXXXXXXX"

```

Now that we have the fact set – in Example 5-23 we called it 'auth_token' – we can perform API operations against our PowerVC environment using Ansible.

Creating VMs with IBM PowerVC using URI module

To create a new VM using the URI module (once you have your authentication key) you have to provide a number of key values, and Unique IDs. These include:

- ▶ The ID of the PowerVC Project you want to deploy the new VM in e.g. 'ibm-default'
- ▶ The ID of the PowerVC Image you want to use for the new VM
- ▶ The ID of the PowerVC compute flavor you want to use for the new VM
- ▶ The ID of the PowerVC network on which to place the new VM

There are a number of other optional values you can supply including availability zone (host group or name of server), key_name (SSH key pair name) and network fixed IP (specific IP). These values are detailed in the [OpenStack API compute \(nova\) documentation](#).

In Example 5-24 we show how to build a new VM using the URI module via IBM PowerVC's API nova service.

Example 5-24 Create a new VM on PowerVC using the URI module and API

```

- name: Connect to PowerVC with token and create a new VM
  uri:
    url: https://{ powervc_host }:{ nova_port }/v2.1/{ project_id }/servers
    method: POST
    use_proxy: no

```

```

validate_certs: no
return_content: no
body: '{
    "server": {
        "name": "{{ new_vm_name }}",
        "imageRef": "{{ image_UID }}",
        "flavorRef": "{{ flavor_UID }}",
        "availability_zone": "{{ host_group_name }}",
        "networks": [
            {
                "uuid": "{{ network_UID }}"
            }
        ]
    }
}'
body_format: json
headers:
    Accept: "application/json"
    Content-Type: "application/json"
    OpenStack-API-Version: "compute 2.46"
    User-Agent: "python-novaclient"
    X-Auth-Token: "{{ auth_token }}"
    X-OpenStack-Nova-API-Version: "2.46"
status_code: 202
register: vm_create

```

Note: In Example 5-24 on page 244 you need to pass the project id, image id, flavor id and network id.

The status code for a successful deploy is '202'.

Destroying VMs with IBM PowerVC using URI module

To destroy a VM using API services from PowerVS we need to know two things:

- ▶ The ID of the PowerVC Project where the VM is hosted e.g. 'ibm-default'
- ▶ The ID of the PowerVC VM you want to destroy

Note: When we created a new VM we passed the VM name to the API, however when deleting an existing VM we have to use the VMs unique ID.

We discuss how to retrieve a project ID in “Collect project ID using the project name from PowerVC using URI module” on page 246.

We discuss how to retrieve a VSI ID in “Collect VM ID using the VSI name from PowerVC using URI module” on page 247.

In Example 5-25 we show how to destroy an existing VM using the URI module via the IBM PowerVC API nova service.

Example 5-25 Destroy an existing VM on PowerVC using the URI module and API

```

- name: Connect to PowerVC with token and destroy a VM
  uri:
    url: https://{{ powervc_host }}:{{ nova_port }}/v2.1/{{ project_id }}/servers/{{
vm_id }}
    method: DELETE
    use_proxy: no
    validate_certs: no
    return_content: no
    headers:

```

```

    X-Auth-Token: "{{ auth_token }}"
    status_code: 204
    register: vm_destroy

```

Note: The status code for a successful VM destroy is '204'

Retrieving resource information from IBM PowerVC using URI module

When using APIs we cannot always refer to names such as project name, VM name, network name etc. when referencing the endpoint. OpenStack and PowerVC APIs work with unique IDs. Although we can see a lot of these IDs from the PowerVC UI or the OpenStack command line we don't expect people developing Ansible playbooks to know them, or hard code them. We therefore have to convert resource names into their resource IDs before we can perform any meaningful operations on PowerVC using APIs.

Collect project ID using the project name from PowerVC using URI module

In Example 5-26 we show the URI module connecting to the PowerVC 'projects' API service to retrieve all project information, using the authorization token collected in Example 5-23 on page 244. We then filter that information to select just the project we are interested in (ibm-default in this case). Finally we set a fact called 'project_id' containing just the ID of our selected project and display that ID.

Example 5-26 Retrieve a PowerVC project ID using URI module

```

- name: Connect to PowerVC with auth token to collect project information
  uri:
    url: https://{{ powervc_host }}:{{ auth_port }}/v3/projects
    method: GET
    use_proxy: no
    validate_certs: no
    return_content: no
    headers:
      X-Auth-Token: "{{ auth_token }}"
    register: project_information

- name: Collect ID of chosen project in array format
  set_fact:
    project_id_array: "{{ project_information.json | json_query(query) }}"
  vars:
    query: "projects[?name=='ibm-default'].{id: id}"

- name: Collect project ID for selected project
  set_fact:
    project_id: "{{ project_id_array.0['id'] }}"

- name: Show Project ID
  debug:
    var: project_id

```

The output from Example 5-26 is shown in Example 5-27.

Example 5-27 Output from using URI module to retrieve a project ID from a project name

```

TASK [Collect ID of chosen project in array format]
*****
ok: [localhost]

TASK [Collect project ID for selected project]
*****

```

```
ok: [localhost]
```

```
TASK [Show Project ID]
```

```
*****
ok: [localhost] => {
  "project_id": "6a01a6c6f13c40f79b7ff5552170a371"
}
```

We can now use that project ID variable in future PowerVC API Ansible playbooks such as creating a new VM.

Collect VM ID using the VSI name from PowerVC using URI module

In Example 5-28 we show the URI module connecting to the PowerVC 'nova' API service to retrieve information about all the VMs (servers), using the authorization token collected in Example 5-26 on page 246 and the project id collected in Example 5-27 on page 246. We then filter that information to select just the VM we are interested in. Finally we set a fact called 'vm_id' containing just the ID of our selected VM and display that ID.

Example 5-28 Retrieve a PowerVC VM ID using URI module

```
- name: Connect to PowerVC with auth token and project ID to collect VM information
  uri:
    url: https://{{ powervc_host }}:{{ nova_port }}/v2.1/{{ project_id }}/servers
    method: GET
    use_proxy: no
    validate_certs: no
    return_content: no
    headers:
      X-Auth-Token: "{{ auth_token }}"
    register: vm_information

- name: Collect ID of chosen VM in array format
  set_fact:
    vm_id_array: "{{ vm_information.json | json_query(query) }}"
  vars:
    query: "servers[?name=='{{ vm_name }}'].{id: id}"

- name: Collect VM ID for selected VM
  set_fact:
    vm_id: "{{ vm_id_array.0['id'] }}"

- name: Show VM ID
  debug:
    var: vm_id
```

Note: In Example 5-28 we have had to use the 'project_id' in the API URL and the VM name has been passed as a variable {{ vm_name }}.

The output from Example 5-28 is shown in Example 5-29.

Example 5-29 Output from using URI module to retrieve a VM ID from a VM name

```
TASK [Collect ID of chosen VM in array format]
```

```
*****
ok: [localhost]
```

```
TASK [Collect VM ID for selected VM]
```

```
*****
```

```
ok: [localhost]
```

```
TASK [Show VM ID]
```

```
*****
ok: [localhost] => {
  "vm_id": "1b9efb52-3b8a-4927-af84-c0feef495c1f"
}
```

We can now use that VM ID variable in future PowerVC API Ansible playbooks such as destroying an existing VM or performing PowerVC operations against that VM.

Resize a VM using URI modules and PowerVC API services

A key advantage of using the PowerVC API services is that we are able to perform more detailed tasks such as resizing a VM. This can be useful if a VM is low on CPU or memory resources and you want to increase them, or when a VM needs to reduce its resources for example after a development test phase.

In this section we introduce the VM ‘action’ API service that allows us to perform a number of different actions against an existing VM including online resizing. The options are documented in the [IBM PowerVC documentation](#). In Example 5-30 we pass the VM action API service the new values for required CPU and memory.

Example 5-30 Resize an active VM using URI module and PowerVC API services

```
- name: "Connect to PowerVC with token and resize VM {{ vm_name }} to {{
new_total_proc_units }} processors, and {{ new_total_memory_mb }}MB"
  uri:
    url: https://{{ powervc_host }}:{{ nova_port }}/v2.1/{{ project_id }}/servers/{{
vm_id }}/action
    method: POST
    use_proxy: no
    validate_certs: no
    return_content: no
    body: {
      "resize": {
        "flavor": {
          "vcpus": "{{ new_vcpus }}",
          "disk": "0",
          "extra_specs": {
            "powervm:proc_units": "{{ new_total_proc_units }}",
          },
          "ram": "{{ new_total_memory_mb }}"
        }
      }
    }
  body_format: json
  headers:
    Accept: "application/json"
    Content-Type: "application/json"
    OpenStack-API-Version: "compute 2.46"
    User-Agent: "python-novaclient"
    X-Auth-Token: "{{ auth_token }}"
    X-OpenStack-Nova-API-Version: "2.46"
  status_code: 202
  register: vm_resize_details
```

The VM we created in “Creating VMs with IBM PowerVC using URI module” on page 244 was assigned 1 vCPU, 0.5 entitled cores and 4GB of memory, as we can see from the PowerVC UI in Figure 5-4.

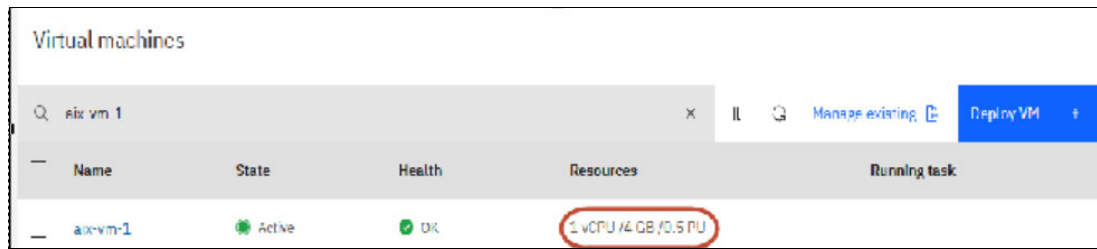


Figure 5-4 PowerVC UI showing VM resource before resize using API services

In Example 5-31 we show the output from the resizing playbook.

Example 5-31 Output showing resize of online VM using URI module and PowerVC API services

```
TASK [Show current CPU and memory allocation for VM aix-vm-1]
*****
ok: [localhost] => {
  "current_vm_spec_details": {
    "CPUs": "0.50",
    "Memory": 4096,
    "name": "aix-vm-1",
    "vCPUs": 1
  }
}

TASK [Connect to PowerVC and resize VM aix-vm-1 to 0.75 processors, and 6144MB]
*****
ok: [localhost]

TASK [Connect to PowerVC with token and wait for VM aix-vm-1 to be in state
'VERIFY_RESIZE'] *****
FAILED - RETRYING: [localhost]: Connect to PowerVC with token and wait for VM aix-vm-1 to
be in state 'VERIFY_RESIZE' (6 retries left).
FAILED - RETRYING: [localhost]: Connect to PowerVC with token and wait for VM aix-vm-1 to
be in state 'VERIFY_RESIZE' (5 retries left).
ok: [localhost]

TASK [Connect to PowerVC with token and confirm resize of VM aix-vm-1]
*****
ok: [localhost]

TASK [Pause for 30 seconds to allow resizing to complete]
*****
Pausing for 30 seconds
(ctrl+C then 'C' = continue early, ctrl+C then 'A' = abort)
ok: [localhost]

TASK [Connect to PowerVC and collect new CPU and memory information for VM aix-vm-1 after
the resize]
*****
ok: [localhost]

TASK [Show new CPU and memory allocation for VM aix-vm-1]
*****
ok: [localhost] => {
```

```

"new_vm_spec_details": {
  "CPUs": "0.75",
  "Memory": 6144,
  "name": "aix-vm-1",
  "vCPUs": 1
}
}

```

```

PLAY RECAP *****
localhost : ok=19   changed=0    unreachable=0    failed=0    skipped=6    rescued=0
ignored=0

```

The output shown in Example 5-31 on page 249 that the VM reported 0.5 CPU entitlement and 4 GB of memory before the resize and 0.75 CPU entitlement and 6 GB of memory after the resize.

We can also see the resize being performed in the PowerVC UI in Figure 5-5

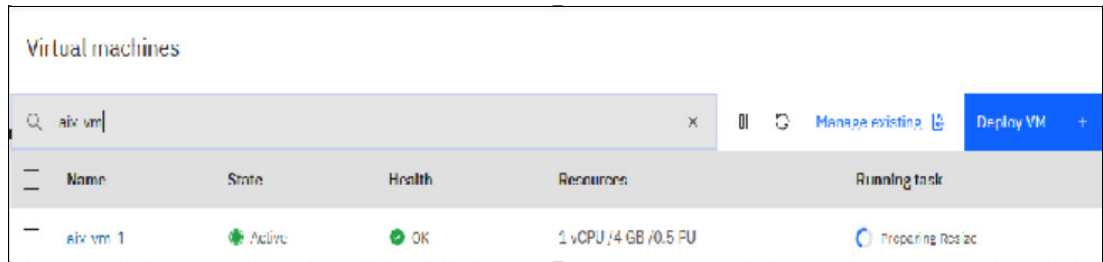


Figure 5-5 PowerVC UI showing VM resizing using API services

Once the resize has completed we can verify on the PowerVC UI as shown in Figure 5-6.

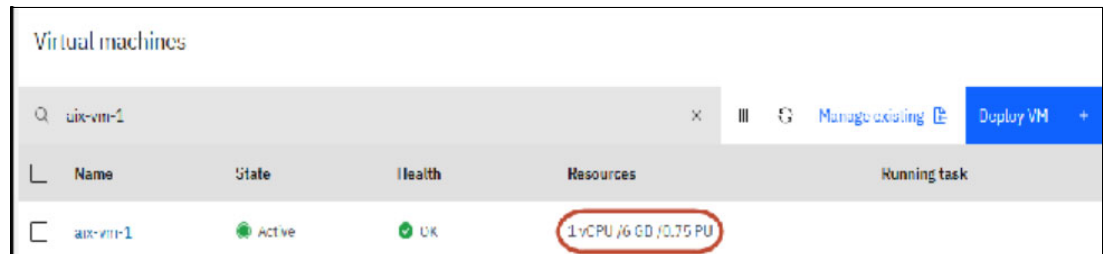


Figure 5-6 PowerVC UI showing VM after resize using API services

5.2 IBM Power Virtual Server (PowerVS)

IBM Power Virtual Server is an Infrastructure as a Service (IaaS) offering that allows customers to deploy AIX, IBM i and Linux workloads in a public cloud environment. The PowerVS data centers are spread across the globe including America, Canada, Brazil, UK, Germany, Japan and Australia.

Note: Within PowerVS, virtual machines (VMs) are referred to as virtual server instances (VSIs).

5.2.1 Using the IBM Cloud collection for PowerVS

IBM has created a collection (`ibm.cloudcollection`) to allow Ansible to interact with IBM Cloud. This collection includes a number of Power Infrastructure (PI) modules for use within IBM PowerVS. The collection is available on Galaxy [here](#).

This collection can be installed as shown in Example 5-32.

Example 5-32 Installing the `ibm.cloudcollection` from Galaxy

```
# ansible-galaxy collection install ibm.cloudcollection
Process install dependency map
Starting collection install process
Installing 'ibm.cloudcollection:1.49.0' to
'/root/.ansible/collections/ansible_collections/ibm/cloudcollection'
```

A number of key PI modules are shown in Table 5-4

Table 5-4 IBM Cloud Collection PI modules

Module name	Function
<code>ibm_pi_catalog_images_info</code>	Collect information about PowerVS catalog images
<code>ibm_pi_cloud_connection</code>	Create, update or destroy an IBM Cloud connection
<code>ibm_pi_cloud_instance_info</code>	Collect information about a PowerVS service instance
<code>ibm_pi_instance</code>	Create, update or destroy a VSI
<code>ibm_pi_instance_action</code>	Perform an action against a VSI
<code>ibm_pi_instance_info</code>	Collect information about a VSI
<code>ibm_pi_instances_info</code>	Collect information about all VSIs
<code>ibm_pi_network</code>	Create, update or destroy a PowerVS network
<code>ibm_pi_volume</code>	Create, update or destroy a PowerVS storage volume
<code>ibm_pi_volume_attach</code>	Attach a PowerVS storage volume to a VSI

In total there are over 70 PowerVS specific modules in the collection.

Note: The IBM Cloud PowerVS modules generate Terraform code to perform the actions against the PowerVS API services. They currently require Terraform v0.10.20 to be installed. The Terraform resources and data sources they call can be found in:

<https://registry.terraform.io/providers/IBM-Cloud/ibm/latest/docs>

Create a new VSI in PowerVS using the IBM Cloud Collection

In this section we will show how to create a new VSI using the `ibm.cloudcollection.ibm.pi_instance` module. When using the IBM Cloud collection we pass the modules the API key, the cloud instance and resource IDs, and the region in each task. An example showing a VSI creation can be seen in Example 5-33 on page 252.

Example 5-33 Create a new VSI using the IBM Cloud Collection module `ibm_pi_instance`

```
- name: Create a POWER Virtual Server Instance
  ibm.cloudcollection.ibm_pi_instance:
    state: available
    pi_cloud_instance_id: "{{ pi_cloud_instance_id }}"
    ibmcloud_api_key: "{{ ibmcloud_api_key }}"
    id: "{{ pi_instance.resource.id | default(omit) }}"
    region: "{{ region }}"
    pi_memory: "{{ memory }}"
    pi_processors: "{{ processors }}"
    pi_instance_name: "{{ vsi_name }}"
    pi_proc_type: "{{ proc_type }}"
    pi_image_id: "{{ image_dict[image_name_to_be_created] }}"
    pi_volume_ids: []
    pi_network_ids:
      - "{{ pi_network.id }}"
    pi_key_pair_name: "{{ pi_ssh_key.pi_key_name }}"
    pi_sys_type: "{{ sys_type }}"
    pi_replication_policy: none
    pi_replication_scheme: suffix
    pi_replicants: "1"
    pi_storage_type: "{{ disk_type }}"
  register: pi_instance_create_output
```

Note: The 'state' option for the `ibm_pi_instance` module to ensure a VSI exists, is 'available'.

Destroy a VSI in PowerVS using the IBM Cloud Collection

To destroy a VSI using the `ibm.cloudcollection.ibm_pi_instance` module you simply need to define the 'state' of that VSI to 'absent' as shown in Example 5-34.

Example 5-34 Destroy a VSI using the IBM cloud collection `ibm_pi_instance` module

```
- name: Destory a POWER Virtual Server Instance
  ibm.cloudcollection.ibm_pi_instance:
    state: absent
    pi_cloud_instance_id: "{{ pi_cloud_instance_id }}"
    ibmcloud_api_key: "{{ ibmcloud_api_key }}"
    id: "{{ pi_instance.resource.id | default(omit) }}"
    region: "{{ region }}"
  register: pi_instance_destroy_output
```

5.2.2 Using the URI module for PowerVS

We are able to use the Ansible URI module to make calls to API services in IBM Cloud PowerVS, similar to OpenStack and PowerVC as described in 5.1.2, "Using the URI modules to interact with PowerVC API services" on page 241.

Just like OpenStack, IBM PowerVS has a large set of API services that allow us to manage resources such as Virtual Server Instances (VSIs), images, storage volumes, key pairs, networks, snapshots, VPNs etc. These APIs are documented in the [IBM Cloud documentation](#).

PowerVS services use regional endpoints over both public and private networks. To target the public service you need to replace `{region}` with the prefix that represents the geographic area where the public facing service is located in the URL shown in Example 5-35 on page 253. Currently these are us-east (Washington DC), us-south (Dallas, Texas), eu-de

(Frankfurt, Germany), lon (London, UK), tor (Toronto, Canada), syd (Sydney, Australia), and tok (Tokyo, Japan).

Example 5-35 Public regional endpoint for IBM PowerVS

```
https://{region}.power-iaas.cloud.ibm.com
```

To target the private service you need to replace `{region}` with the prefix that represents the geographic area where the private facing service is located in the URL shown in Example 5-36. Currently these are us-east (Washington DC), us-south (Dallas, Texas), eu-de (Frankfurt, Germany), eu-gb (London, UK), ca-tor (Toronto, Canada), au-syd (Sydney, Australia), jp-tok (Tokyo, Japan), jp-osa (Osaka, Japan), br-sao (Sao Paolo, Brazil), and ca-mon (Montreal, Canada).

Example 5-36 Private regional endpoint for IBM PowerVS

```
https://private.{region}.power-iaas.cloud.ibm.com
```

All the IBM Cloud PowerVS API methods are also documented, along with the API service URL, the required parameters and the response body. For example, to obtain information about all the VSIs, the request is documented at:

<https://cloud.ibm.com/apidocs/power-cloud#pcloud-pvminstances-getall>

Example 5-37 shows an example request to retrieve all VSIs within IBM PowerVS.

Example 5-37 Example request to get all PowerVS VSI information

```
curl -X GET
https://{region}.power-iaas.cloud.ibm.com/pcloud/v1/cloud-instances/${CLOUD_INSTANCE_ID}/pv
m-instances
-H 'Authorization: Bearer <>'
-H 'CRN: crn:v1...'
-H 'Content-Type: application/json'
```

Authenticating with PowerVS using APIs

Before we can perform any action against the API services presented by IBM PowerVS we must first authenticate. To do this we require an IBM Cloud API key which will allow us to obtain a Cloud IAM access token. This IAM access token (often referred to as ‘auth token’) can then be used to directly access the API services.

IBM Cloud API keys are associated with a user's identity and can be used to access cloud platform and APIs, depending on the access that is assigned to the user. The API access key is created using <https://cloud.ibm.com/iam/apikeys>.

Note: When creating an IBM Cloud API key record the key in a safe location as you will be unable to retrieve the contents after it has been created.

In Example 5-38 on page 254 we show how to obtain the auth token using the URI module along with the IBM Cloud API key.

Example 5-38 Obtain the IAM access token (auth token) using URI module and IBM Cloud API key

```

- name: Obtain IBM Cloud PowerVS authorization token using IBM Cloud API key
  hosts: localhost
  gather_facts: no

  vars:
    - auth_data: "grant_type=urn:ibm:params:oauth:grant-type:apikey&apikey="
      api_key: "xxxxxxxxxxxxxxxxxxxxxxxxxxxx"

  tasks:
    - name: Get IAM access token
      uri:
        url: "https://iam.cloud.ibm.com/identity/token"
        method: POST
        force_basic_auth: true
        validate_certs: yes
        headers:
          content-type: "application/x-www-form-urlencoded"
          accept: "application/json"
        body: "{{ auth_data }}{{ api_key|trim }}"
        body_format: json
        register: iam_token_request

    - name: Set auth token fact
      set_fact:
        auth_token: "{{ iam_token_request.json.access_token }}"

    - name: Show token
      debug:
        var: auth_token

```

In Example 5-39 we point to the IBM [Cloud identity](#) and in the body of the API POST we pass two values:

- ▶ Authorization Data (grant_type=urn:ibm:params:oauth:grant-type:apikey&apikey=)
- ▶ IBM API key (trimmed)

The last task in the playbook outputs the 'auth_token' fact, which we populated. Normally this would be hidden but we have included it so we can confirm it has been created correctly.

Example 5-39 Output from IAM access token retrieval

```

PLAY [Obtain IBM Cloud PowerVC auth token using API]
*****

TASK [Get IAM acces token]
*****
ok: [localhost]

TASK [Show token]
*****
ok: [localhost] => {
  "auth_token": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
}

```

This 'auth_token' can now be used within the Ansible playbook to perform actions against the IBM PowerVS API services.

Before we perform an action against the IBM PowerVS API services associated with our IBM Power resources, we need to know the Power Systems Virtual Server Instance ID (also known as the Cloud Resource Name or CRN). We can obtain our CRN using the `ibmcloud` command line or via the IBM Cloud UI as shown in Example 5-40.

Example 5-40 Obtain our Cloud Resource Name (CRN) via the `ibmcloud` command line

```
% ibmcloud resource service-instance "Power Virtual Server-London 06" --id
Retrieving service instance Power Virtual Server-London 06 in all resource groups under
account XXX YYYYY's Account as x_yyyyy@uk.ibm.com...
crn:v1:bluemix:public:power-iaas:lon06:a/abcdefghijklmnopqrstuvwxyabcdefghijklmnop:121d5ee5-b87d-4a0
e-86b8-aaff422135478::
```

In Example 5-40 the CRN includes:

- ▶ Tenant ID - in the example above: `abcdefghijklmnopqrstuvwxyabcdefghijklmnop`
- ▶ Cloud Instance ID - in the example above: `121d5ee5-b87d-4a0e-86b8-aaff422135478`

As well as the CRN tenant ID and cloud instance ID shown above, we also have to define the following CRN values:

Example 5-41 CRN values required to connect to PowerVS London 04 API services

```
crn:
  version: "v1"
  cname: "bluemix"
  ctype: "public"
  service_name: "power-iaas"
  location: "lon04"
  tenant_id: "abcdefghijklmnopqrstuvwxyabcdefghijklmnop"
  cloud_instance_id: "121d5ee5-b87d-4a0e-86b8-aaff422135478"
```

Note: In the CRN values shown in Example 5-42, tenant id and cloud instance id are those collected previously in Example 5-41 on page 255. **Location** must be one of the locations listed in the `ibmcloud catalog locations` command line output e.g. `fra01`, `fra02`, `lon04`, `lon06`, `dal10`, `dal12`, `wdc06`, `wdc07`, `mon01`, `tor01`, `osa21`, `sao01`, `sao04`, `syd04`, `syd05` and `tok04`

Retrieving resource information from IBM PowerVS using URI module

In this section we will show examples of how to retrieve information about resources such as VSIs, images and networks from IBM PowerVS using URI and API servers.

Retrieve information about all VSIs

In Example 5-42 we retrieve the names of all existing VSIs in our PowerVS workspace using the CRN values defined, along with the GET method and URL documented in Example 5-38 on page 254.

Example 5-42 Retrieve all VSI names in PowerVS using URI module and APIs

```
- name: Collect information about all the VSI's in this cloud instance
  uri:
    url: "https://{{ region }}.power-iaas.cloud.ibm.com/pcloud/{{ api_version
}}/cloud-instances/{{ crn.cloud_instance_id }}/pvm-instances"
    method: GET
    headers:
      Authorization: "Bearer {{ auth_token }}"
      CRN: "crn:{{ crn.version }}:{{ crn.cname }}:{{ crn.ctype }}:{{ crn.service_name
}}:{{ crn.location }}:a/{{ crn.tenant_id }}:{{ crn.cloud_instance_id }}::"
```

```

    Content-Type: application/json
    register: pvs_existing_vsi_results

- name: Set VSI list of names
  set_fact:
    vsi_names: "{{ vsi_names | default([]) + [item] }}"
  with_items: "{{ pvs_existing_vsi_results | json_query(query_to_run) }}"
  vars:
    query_to_run: 'json.pvmInstances[*].serverName'

```

We can see the output of our VSI retrieval request in Example 5-43.

Example 5-43 Display the names of all PowerVS VSIs using URI module and APIs

```

TASK [Collect information about all the VSI's in this cloud instance]
*****
ok: [localhost]

TASK [Set VSI list of names]
*****
*****
ok: [localhost] => (item=aix-vsi-1)
ok: [localhost] => (item=ibmi-vsi-1)

```

Retrieve information about all images

In Example 5-44 we retrieve the names of all existing images in our PowerVS catalog. The API syntax is documented can be found at:

<https://cloud.ibm.com/apidocs/power-cloud#pcloud-cloudinstances-images-getall>

Example 5-44 Retrieve all VSI images within our PowerVS environment using URI module and APIs

```

- name: Collect information about all the images in this cloud instance
  uri:
    url: "https://{{ region }}.power-iaas.cloud.ibm.com/pcloud/{{ api_version
}}/cloud-instances/{{ crn.cloud_instance_id }}/images"
    method: GET
    headers:
      Authorization: "Bearer {{ auth_token }}"
      CRN: "crn:{{ crn.version }}:{{ crn.cname }}:{{ crn.ctype }}:{{ crn.service_name
}}:{{ crn.location }}:a/{{ crn.tenant_id }}:{{ crn.cloud_instance_id }}::"
      Content-Type: application/json
    register: pvs_images_results

- name: Set list of image names
  set_fact:
    image_names: "{{ image_names | default([]) + [item] }}"
  with_items: "{{ pvs_images_results | json_query(query_to_run) }}"
  vars:
    query_to_run: 'json.images[*].name'

- name: Show all image names
  debug:
    var: image_names

```

The output of the image retrieval can be seen in Example 5-45

Example 5-45 Show all images within our PowerVS environment

```
TASK [Set VSI list of names]
*****
ok: [localhost] => (item=7300-01-01)
ok: [localhost] => (item=IBMi-75-01-2984-2)

TASK [Show all image names]
*****
ok: [localhost] => {
  "image_names": [
    "7300-01-01",
    "IBMi-75-01-2984-2"
  ]
}
```

Retrieve information about all networks

In Example 5-46 we retrieve the names of all existing images in our PowerVS environment. The API syntax is documented at:

<https://cloud.ibm.com/apidocs/power-cloud#pcloud-cloudinstances-networks-getall>

Example 5-46 Retrieve all networks within our PowerVS environment using URI module and APIs

```
- name: Collect information about all the networks within PowerVS environment
  uri:
    url: "https://{{ region }}.power-iaas.cloud.ibm.com/pcloud/{{ api_version
}}/cloud-instances/{{ crn.cloud_instance_id }}/networks"
    method: GET
    headers:
      Authorization: "Bearer {{ auth_token }}"
      CRN: "crn:{{ crn.version }}:{{ crn.cname }}:{{ crn.ctype }}:{{ crn.service_name
}}:{{ crn.location }}:a/{{ crn.tenant_id }}:{{ crn.cloud_instance_id }}::"
      Content-Type: application/json
    register: pvs_network_results

- name: Set network of names
  set_fact:
    network_names: "{{ network_names | default([]) + [item] }}"
  with_items: "{{ pvs_network_results | json_query(query_to_run) }}"
  vars:
    query_to_run: 'json.networks[*].name'

- name: Show all network names
  debug:
    var: network_names
```

The output of the network retrieval can be seen in Example 5-47 on page 258.

Example 5-47 Show all networks within our PowerVS environment

```
TASK [Set network of names]
*****
ok: [localhost] => (item=public-192_168_151_128-29-VLAN_2044)
ok: [localhost] => (item=private-subnet2)
ok: [localhost] => (item=private-subnet1)

TASK [Show all network names]
*****
ok: [localhost] => {
  "network_names": [
    "public-192_168_151_128-29-VLAN_2044",
    "private-subnet2",
    "private-subnet1"
  ]
}
```

Retrieve all information about a specific resource in PowerVS

As with OpenStack, when using IBM PowerVS APIs we cannot always refer to names such as VSI name, image name, network name etc. when referencing the endpoint. When we want to perform an action such as against a specific resource, we have to refer to the resource's unique ID. Therefore we must first retrieve that ID before we can reference it. For example, if we want to show all the details about a specific VSI, we need to provide the ID of that VSI. In order to do this, we have to convert the VSI name (which we know) into its ID.

In Example 5-48 we use the URI module and the PowerVS 'pvm-instances' API service to retrieve information about all the VSIs. We then filter that information and collect just the ID of the VM we are interested in, `{{ vsi_name }}`.

Example 5-48 Retrieve the ID of a PowerVS VSI using its name

```
- name: Collect information about all the VSI's in this cloud instance
  uri:
    url: "https://{{ region }}.power-iaas.cloud.ibm.com/pcloud/{{ api_version }}/cloud-instances/{{ crn.cloud_instance_id }}/pvm-instances"
    method: GET
    headers:
      Authorization: "Bearer {{ auth_token }}"
      CRN: "crn:{{ crn.version }}:{{ crn.cname }}:{{ crn.ctype }}:{{ crn.service_name }}:{{ crn.location }}:a/{{ crn.tenant_id }}:{{ crn.cloud_instance_id }}::"
      Content-Type: application/json
    register: pvs_existing_vsi_results

- name: Collect ID of chosen VSI in array format
  set_fact:
    vsi_id_array: "{{ pvs_existing_vsi_results.json | json_query(query) }}"
  vars:
    query: "pvmInstances[?serverName=='{{ vsi_name }}'].{id: pvmInstanceID}"

- name: Collect VSI ID for selected VM
  set_fact:
    vsi_id: "{{ vsi_id_array.0['id'] }}"

- name: Show VSI ID
  debug:
    msg: "ID for {{ vsi_name }} is: {{ vsi_id }}"
```

Note: The VSI ID is called 'pvmlInstanceID' in PowerVS.

The output from the VSI ID retrieval can be seen in Example 5-49

Example 5-49 Show the VSI ID

```
TASK [Show VSI ID]
*****
ok: [localhost] => {
  "msg": "ID for ibmi-vsi-1 is: 2d8bf009-5922-40f4-9eef-d06fec7xxxxx"
}
```

Create a new VSI in IBM PowerVS using URI module and API services

In this section we will create a new PowerVS VSI using the authorization token, along with the image name and network name collected in the previous sections.

The syntax required to create a new VSI using a POST HTTP method is documented at: <https://cloud.ibm.com/apidocs/power-cloud#pcloud-pvminstances-post>

In Example 5-24 on page 244 we showed an example where the content of the URI POST was contained within the body. In Example 5-50 we define a variable called `{{ vsi_info }}` which contains all the required information such as VSI name, image, network etc. We then use that variable in the body section of the URI module.

Example 5-50 Create a new VSI within PowerVS using URI module and API services

Variable definition

```
vsi_info:
  serverName: "aix-vsi-1"
  imageID: "7300-01-01"
  processors: 1
  procType: "shared"
  memory: 4
  sysType: "s922"
  storageType: "tier3"
  networkIDs:
    - "public-192_168_xxx_xxx-VLAN_2044"
```

Create VSI task

```
- name: Create new VSI
  uri:
    url: "https://{{ region }}.power-iaas.cloud.ibm.com/pcloud/{{ api_version }}/cloud-instances/{{ crn.cloud_instance_id }}/pvm-instances"
    method: POST
    status_code: 201
    body_format: json
    body: "{{ vsi_info | to_json }}"
    headers:
      Authorization: "Bearer {{ auth_token }}"
      CRN: "crn:{{ crn.version }}:{{ crn.cname }}:{{ crn.ctype }}:{{ crn.service_name }}:{{ crn.location }}:a/{{ crn.tenant_id }}:{{ crn.cloud_instance_id }}:."
      Content-Type: application/json
  register: pvs_create_vsi_result
```

Note: Status code for a successful VSI creation is 201

We can see the VSI being built on the PowerVS UI in Figure 5-7.

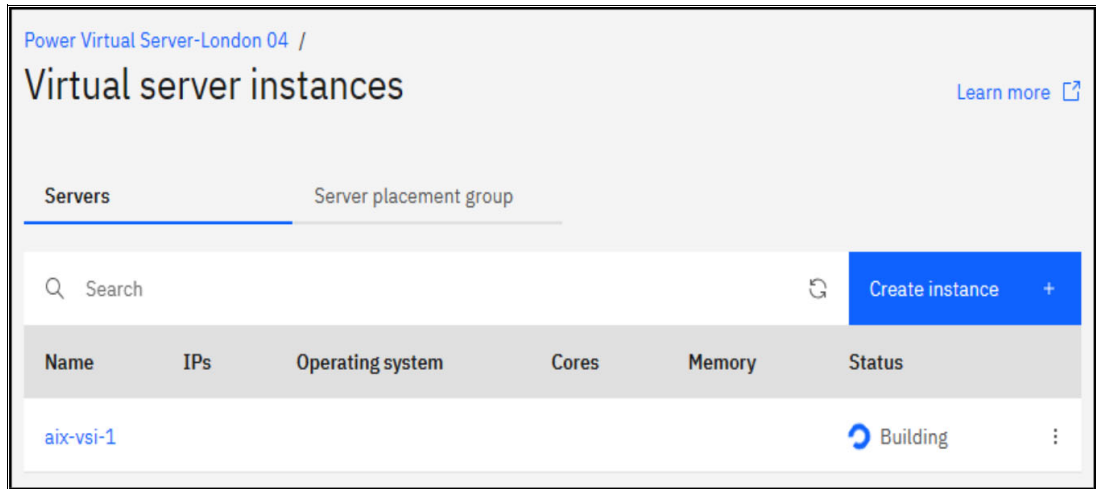


Figure 5-7 PowerVS UI - VSI building

Destroy a VSI in IBM PowerVS using URI module and API services

In this section we will destroy a existing PowerVS VSI using the authorization token and the VSI name. To destroy an existing VSI you are required to pass its ID, along with the DELETE HTTP method. As we only know the VSI name, we have to obtain its ID first.

The syntax required to destroy a VSI using a DELETE HTTP method is documented at: <https://cloud.ibm.com/apidocs/power-cloud#pcloud-pvminstances-delete>

In Example 5-51 we show how to destroy a VSI in PowerVS by passing the VSI name – {{ vsi_name }}. This is then converted into the VSI ID which is used for the deletion command.

Example 5-51 Destroy a PowerVS VSI using URI module and API

```
- name: Collect information about all the VSI's in this cloud instance
  uri:
    url: "https://{{ region }}.power-iaas.cloud.ibm.com/pcloud/{{ api_version
}}/cloud-instances/{{ crn.cloud_instance_id }}/pvm-instances"
    method: GET
    headers:
      Authorization: "Bearer {{ auth_token }}"
      CRN: "crn:{{ crn.version }}:{{ crn.cname }}:{{ crn.ctype }}:{{ crn.service_name
}}:{{ crn.location }}:a/{{ crn.tenant_id }}:{{
crn.cloud_instance_id }}::"
      Content-Type: application/json
    register: pvs_existing_vsi_results

- name: SHow all
  debug:
    var: pvs_existing_vsi_results

- name: Collect ID of chosen VSI in array format
  set_fact:
    vsi_id_array: "{{ pvs_existing_vsi_results.json | json_query(query) }}"
  vars:
    query: "pvmInstances[?serverName=='{{ vsi_name }}'].{id: pvmInstanceID}"

- name: Collect VSI ID for selected VM
  set_fact:
    vsi_id: "{{ vsi_id_array.0['id'] }}"
```

```

- name: Show details of VSI to destroy
  debug:
    msg: "Destroying VSI name: {{ vsi_name }} ID: {{ vsi_id }}"

- name: Destroying VSI
  uri:
    url: "https://{{ region }}.power-iaas.cloud.ibm.com/pcloud/{{ api_version
}}/cloud-instances/{{ crn.cloud_instance_id }}/pvm-instances/{{ vsi_id }}"
    method: DELETE
    status_code: 200
    body_format: json
    #body: "{{ vsi_name }}"
    headers:
      Authorization: "Bearer {{ auth_token }}"
      CRN: "crn:{{ crn.version }}:{{ crn.cname }}:{{ crn.ctype }}:{{ crn.service_name
}}:{{ crn.location }}:a/{{ crn.tenant_id }}:{{ crn.cloud_instance_id }}::Content-Type:
application/json
    register: pvs_destroy_vsi_result

```

The output from running the playbook shown in Example 5-51 on page 260 is shown in Example 5-52 where the name and ID of the VSI are displayed before destroying it.

Example 5-52 Output of destroying VSI in PowerVS using URI module and API

```

TASK [Collect ID of chosen VSI in array format]
*****
ok: [localhost]

TASK [Collect VSI ID for selected VM]
*****
ok: [localhost]

TASK [Show details of VSI to destroy]
*****
ok: [localhost] => {
  "msg": "Destroying VSI name: aix-vsi-1 ID: d53fbe08-a613-41de-9016-047xxxxx"
}

TASK [Destroying VSI]
*****
ok: [localhost]

```

We can see the VSI delete tasks recorded in the event logs on the PowerVS UI in Figure 5-8.

Resource type	Action	Date	Severity
Virtual Server Instance	Delete	8/22/2023, 6:45:47 AM	Info
Message: Power virtual server instance 'aix-vsi-1' has been deleted. User ID: IBMid-1100008F33 User email:			

Figure 5-8 PowerVS UI Event log showing VSI deletion.

Resizing a VSI on PowerVS using APIs

Using the API services in PowerVS we are able to resize an active VM using the URI module within Ansible. This can be very useful if resources become constrained on a VSI or if we want to reduce resources to save operational costs. Previously, in the section “Create a new VSI in IBM PowerVS using URI module and API services” on page 259, we built a new VSI with 0.5 cores and 4GB of memory, as can be seen in Figure 5-9.

Name	IPs	Operating system	Cores	Memory	Status
aix-vsi-1	192.168.1.100	AIX	0.5 cores	4 GiB	Active

Figure 5-9 IBM PowerVS prior to API resize

In Example 5-53 we show an example of using the URI module and API services to resize that VSI to 0.75 cores and 6GB of memory.

Example 5-53 Resize a PowerVS VSI using URI module and API services

```
- name: "Resize VSI {{ vsi_name }} to 0.75 cores and 6GB of memory"
  uri:
    url: "https://{{ region }}.power-iaas.cloud.ibm.com/pcloud/{{ api_version
}}/cloud-instances/{{ crn.cloud_instance_id }}/pvm-instances/{{ vsi_id }}"
    method: PUT
    status_code: 202
    body_format: json
    body: '{
      "processors": 0.75,
      "memory": 6
    }'
    headers:
      Authorization: "Bearer {{ auth_token }}"
      CRN: "crn:{{ crn.version }}:{{ crn.cname }}:{{ crn.ctype }}:{{ crn.service_name
}}:{{ crn.location }}:a/{{ crn.tenant_id }}:{{ crn.cloud_instance_id }}:."
      Content-Type: application/json
    register: vsi_resize_details
```

During the resize, we can see the status change on the PowerVS UI, as shown in Figure 5-10.

Name	IPs	Operating system	Cores	Memory	Status
aix-vsi-1	192.168.1.100	AIX	0.5 cores	4 GiB	Resize

Figure 5-10 PowerVS VSI resize

The output from the playbook also shows us the resize taking place as seen in Example 5-54.

Example 5-54 Output of PowerVS VSI resize using API services

```
TASK [Show existing CPU and memory allocation for VSI aix-vsi-1]
*****
ok: [localhost] => {
  "current_vsi_spec_details": {
    "CPUs": 0.5,
    "Memory": 4,
    "name": "aix-vsi-1",
    "vCPUs": 1
```

```

    }
}

TASK [Resize VSI aix-vsi-1 to 0.75 cores and 6GB of memory]
*****
ok: [localhost]

TASK [Sleep to allow resize to complete]
*****
Pausing for 180 seconds
(ctrl+C then 'C' = continue early, ctrl+C then 'A' = abort)
ok: [localhost]

TASK [Show new CPU and memory allocation for VSI aix-vsi-1]
*****
ok: [localhost] => {
  "new_vsi_spec_details": {
    "CPUs": 0.75,
    "Memory": 6,
    "name": "aix-vsi-1",
    "vCPUs": 1
  }
}
PLAY RECAP *****
localhost : ok=16  changed=0  unreachable=0  failed=0  skipped=2  rescued=0
ignored=0

```

The output from the PowerVS UI confirms the resize was successful as shown in Figure 5-11.

Name	IPs	Operating system	Cores	Memory	Status
aix-vsi-1	192.168.1.100	AIX	0.75 cores	6 GiB	Active

Figure 5-11 PowerVS UI showing VSI post resize using API services

**6**

Day 2 Management Operations

There is a lot of discussion about “Day 2 operations”, but what does that mean exactly? This chapter defines what Day 2 operations are, and describes how you can use Ansible as a tool to help you automate those operations in your IBM Power environment, whether you are running Linux, AIX or IBM i environments, and show you how you can to optimize managing your hardware and software after the initial installation is done.

The following topics are covered in this chapter:

- ▶ Introduction to Day 2 operations
- ▶ Day 2 operations in Linux servers
- ▶ Day 2 operations in AIX environments
- ▶ Day 2 operations in IBM i environments

6.1 Introduction to Day 2 operations

Day 2 operations are a group of tasks that monitor and maintain your environment after the initial installation. To operationalize a given technology, enterprises need to understand how it will fit into their broader architecture. Leaders must consider both their technical and operational architecture. After all, all systems are made up of not only software but also people and processes.

There are three stages of operations: Day 0, Day 1, and Day 2.

- **Day 0** is the “**design**” stage, where we figure out what resources are required to provide the necessary functions for your information technology project.
- **Day 1** operations describe the “**deployment**” stage, where we actually install, set up, and configure your environment.
- **Day 2** is the “**maintenance**” stage. Typical Day 2 operations are focused around maintaining, monitoring, and optimizing the system. Day 2 operations continue throughout the product lifecycle, as the system behavior must be continuously analyzed and patched.

In addition to monitoring how your environment is running, Day 2 operations also entail routine tasks such as installing upgrades and updating systems.

Day 2 operations essentially are involved in maintaining the products/platforms. They continuously monitor the health of the system, validate that it is meeting business requirements, track and fix issues that arise, and validate that all required patches and updates are applied to keep the environment secure. Since most of the Day 2 operations are continuous and repetitive activities, they often are tasks that should be considered for automation using Ansible. For more information on how Red Hat supports Day 2 operations see this [Red Hat Document](#).

In the following sections we discuss using Ansible automation in your IBM Power environment – whether using IBM AIX, IBM i, or Linux on Power – to assist you in managing the following functions within your environment:

- ▶ Storage
- ▶ Security and compliance
- ▶ Fixes or Upgrades
- ▶ Configuration and Tuning

Storage

The storage tasks discuss how to manage and maintain how your data is stored in your servers. This involves things like monitoring file systems to ensure that they do not run out of space, monitoring the performance of the storage to ensure it is meeting business requirements, and generally managing a logical volume manager (LVM) and local filesystems (FS) at the operating systems (OS) level. Using Ansible, you can automate the storage related tasks with playbooks that are already available for your operating system, or create your own playbooks accordingly. Some of the functions your Ansible playbooks can do are:

- ▶ Create a file system
- ▶ Remove a file system
- ▶ Mount a file system
- ▶ Unmount a file system
- ▶ Create LVM volume groups

- ▶ Remove LVM volume groups
- ▶ Create logical volumes
- ▶ Remove logical volumes

Security and compliance

Each operating system provides security technologies to combat vulnerabilities, protect data, and meet regulatory compliance. Depending on your location, industry, and the entities that you engage with, you may have different data protection regulations that you must adhere to. Some examples of these regulations and standards are:

- ▶ Department of Defense (DoD)
- ▶ General Data Protection Regulation (GDPR)
- ▶ PCI DSS (Payment Card Industry Data Security Standard)
- ▶ Defense Information Systems Agency (DISA) Security Technical Implementation Guide (STIG).
- ▶ Criminal Justice Information Services (CJIS) security policy.
- ▶ Commercial cloud services (C2S).
- ▶ Center for Internet Security (CIS)
- ▶ Health Insurance Portability and Accountability Act (HIPAA).
- ▶ NIST 800-171.
- ▶ Operating System Protection Profile (OSPP) v4.2.
- ▶ Red Hat Corporate Profile for Certified Cloud Providers.

Organizations that you work with may require companies in their supply chain to prove compliance using an independent third-party validation exercise. Ansible Automation Platform can be an optimal solution for an organization to automate regulatory compliance, security configuration and remediation across systems and within containers. An organization can use existing playbook roles that are already available in a community repository or they can develop their own playbooks and roles to meet their specific business requirements.

There are two roles that need to be considered when you design your security playbooks. These roles can be run in separate playbooks or they can be combined into a single playbook. These roles are:

- scanning playbook roles: This role is designed to scan systems based on the requirements set by the business and generate a report file as both a list of system updates required in your systems, and a proof of compliance.
- remediation playbook roles: This role applies the appropriate system setting and applies the required changes to the systems based on their business requirements or industry or governmental requirements for each operating system.

Fixes or Upgrades

To keep your system up-to-date involves:

- planning and configuring how and when security updates are installed
- applying changes introduced by newly updated packages or filesets
- keeping track of security advisories.

As security vulnerabilities are discovered, the affected software must be updated in order to limit any potential security risks. Keeping your system up-to-date requires a patch management solution to manage and install updates. Updates can fix issues that have been

discovered, improve the performance of existing features, or add new features to software. Fixes and patch management solutions for each of the supported operating systems for IBM Power are discussed in the OS related sections later in this chapter.

Configuration and Tuning

Configuration management is a process for maintaining computer systems, servers, applications, network devices, and other IT components to ensure they operate in a desired state. It is a way to help ensure that a system performs as expected, even after changes in the environment occur over time. The configuration management could include all the above activities and help teams to:

- Classify and manage systems by groups and subgroups.
- Centrally modify base configurations.
- Roll out new settings to all applicable systems.
- Automate system identification, patches, and updates.
- Identify outdated, poorly performing, and non-compliant configurations.
- Prioritize necessary actions.
- Access and apply prescriptive remediation.

Due to the scale and complexity of most enterprise environments, IT teams now use automation to define and maintain the desired state of their various systems. For more information see the Red Hat documentation on [Configuration Management](#).

6.2 Day 2 operations in Linux servers

Managing and maintaining your Linux servers can be time consuming and manpower intensive as the number of server images continues to grow. It is important to look at how an automation solution involving Ansible can help you keep up with the growing workload and make your system administrators more efficient. Automation simplifies the management process and can help to eliminate human errors as your staff is doing a large number of repetitive tasks to maintain your server configuration and manage the required security patches and fixes.

Installing system roles for Ansible automation

We can use a role defined for Red Hat Enterprise Linux automation – `rhel-system-roles` – that is a collection of Ansible roles and modules designed to provide a stable and consistent configuration interface to automate and manage Red Hat Enterprise Linux across multiple releases. The effort is based on development of the [Linux System Roles](#) upstream project. There are also SAP related system roles provided by the [SAP LinuxLab](#) upstream project. The use of these roles is discussed in this [Red Hat document](#) and also in section 4.7, “SAP automation” on page 212.

For more details on the Red Hat Enterprise Linux (RHEL) System Roles refer to [this link](#).

The first step is to install the `rhel-system-roles` package on the Ansible controller node. This is done using the following command:

```
yum install rhel-system-roles -y
```

Note: The blivet API packages are also needed which is the python interface that can be used to create scripts for use with administration. The blivet API can be installed with the yum command. The needed packages are blivet-data and python3-blivet.

<https://access.redhat.com/solutions/3776171>

<https://access.redhat.com/solutions/3776211>

6.2.1 Storage

In this section we show some simple tasks as examples of things that you might want to automate in your storage environment on your Linux on IBM Power VMs.

Create a new file system and lv using the RHEL System Role for storage

In this section we will create an Ansible playbook that will allow us to create a new file system in your RHEL virtual machine. To do this follow the steps below:

1. Create a playbook using the RHEL System Role called *rhel-system-roles.storage*. This is shown in Example 6-1.

Example 6-1 Create create_lvm_filesystem_playbook1.yaml playbook

```
# mkdir storage

# cat create_lvm_filesystem_playbook1.yaml
---
- hosts: all
  vars:
    storage_pools:
      - name: myvg
        disks:
          - /dev/mapper/360050768108201d83800000000008e08p1
          - /dev/mapper/360050768108201d83800000000008e08p2
        volumes:
          - name: mylv1
            size: 1 GiB
            fs_type: xfs
            mount_point: /opt/mount1
  roles:
    - rhel-system-roles.storage
```

The simple playbook shown in Example 6-1 does the following:

- Defines a *myvg* volume group which we want to contain the following disks:
 - /dev/mapper/360050768108201d83800000000008e08p1
 - /dev/mapper/360050768108201d83800000000008e08p2
- Defines a logical volume that we want to exist – *mylv1*
- If the *myvg* volume group already exists, the playbook adds the logical volume *mylv1* to the volume group.
- If the *myvg* volume group does not exist, the playbook creates it.
- The playbook then creates an *xf*s file system on the *mylv1* logical volume and persistently mounts the file system at */opt/mount1*.

2. Check the inventory file for the list of target systems and then run the play book created in Example 6-1 on page 269. The inventory file used in this example is the *hosts* file. The process is shown in Example 6-2.

Example 6-2 Run create_lvm_filesystem_playbook1.yaml playbook

```
# pwd
/root/storage

# ls -la
total 52
-rw-r--r--. 1 root root 39216 Aug 27 15:30 ansible.cfg
-rw-r--r--. 1 root root 298 Aug 27 15:35 create_lvm_filesystem_playbook1.yaml
-rw-r--r--. 1 root root16 Aug 27 15:30 hosts
-rw-r--r--. 1 root root 319 Aug 27 15:54 resize_lvm_filesystem_playbook1.yaml
-rw-r--r--. 1 root root 320 Aug 27 16:14 resize_lvm_filesystem_playbook2.yaml

# cat hosts
bs-rbk-lnx-1.power-iaas.cloud.ibm.com

# ansible-playbook create_lvm_filesystem_playbook1.yaml
TASK [rhel-system-roles.storage : Set the list of pools for test verification]
*****
ok: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]

TASK [rhel-system-roles.storage : Set the list of volumes for test
verification] *****
ok: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]

TASK [rhel-system-roles.storage : Remove obsolete mounts]
*****
skipping: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]

TASK [rhel-system-roles.storage : Tell systemd to refresh its view of
/etc/fstab] *****
ok: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]

TASK [rhel-system-roles.storage : Set up new/current mounts]
*****
changed: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com] => (item={'src':
'/dev/mapper/myvg-my1v1', 'path': '/opt/mount1', 'fstype': 'xfs', 'opts':
'defaults', 'dump': 0, 'passno': 0, 'state': 'mounted'})

TASK [rhel-system-roles.storage : Tell systemd to refresh its view of
/etc/fstab] *****
ok: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]

TASK [rhel-system-roles.storage : Retrieve facts for the /etc/crypttab file]
*****
ok: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]

TASK [rhel-system-roles.storage : Manage /etc/crypttab to account for changes
we just made] *****
skipping: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]
```

```
TASK [rhel-system-roles.storage : Update facts]
*****
*****
ok: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]

PLAY RECAP
*****
*****
bs-rbk-lnx-1.power-iaas.cloud.ibm.com : ok=21  changed=3  unreachable=0
failed=0  skipped=11  rescued=0  ignored=0
```

3. Verify the storage configuration using the commands shown in Example 6-3.

Example 6-3 Verify the storage configuration

```
# vgs
VG   #PV #LV #SN Attr   VSize VFree
myvg  2   1   0 wz--n- 19.99g 18.99g

# lvs
LV   VG   Attr      LSize Pool Origin Data%  Meta%  Move Log Cpy%Sync
Convert
mylv1 myvg -wi-ao---- 1.00g

# grep mylv1 /etc/fstab
/dev/mapper/myvg-mylv1 /opt/mount1 xfs defaults 0 0
# df -h |grep '/opt/mount1'
/dev/mapper/myvg-mylv1                1014M   40M  975M   4%
/opt/mount1
```

As we have verified in Example 6-3, the playbook has created a new volume group and logical volume. It also created an *xfs* file system and persistently mounted it as */opt/mount1* directory.

Resize an existing LVM file system using the RHEL System Role for storage

In this section we will create an Ansible playbook that will resize an existing LVM based file system. The first step is to extend the existing volume group which was created in Example 6-2 on page 270.

Note: Only attempt one change at a time. That is, don't try to extend the volume group together with resizing the existing file system in a single playbook.

Extend existing volume group:

To extend the existing volume group, follow the following steps:

1. Add a new disk in the existing volume group to provide space to extend the filesystem. This is shown in Example 6-4.

Example 6-4 To create `resize_lvm_filesystem_playbook1.yaml` playbook

```
# cat resize_lvm_filesystem_playbook1.yaml
---
- hosts: all
  vars:
```

```

storage_pools:
  - name: myvg
    disks:
      - /dev/mapper/360050768108201d83800000000008e08p1
      - /dev/mapper/360050768108201d83800000000008e08p2
      - /dev/mapper/360050768108201d83800000000008e08p2
    volumes:
      - name: mylv1
        size: 1 GiB
        fs_type: xfs
        mount_point: /opt/mount1
roles:
  - rhel-system-roles.storage

```

- Copy some files to the `/opt/mount1` mount point to validate that it is available and then run the second playbook as shown in Example 6-5.

Example 6-5 Example 6-6 Run the `resize_lvm_filesystem_playbook1.yaml` playbook

```

# cp /etc/fstab /opt/mount1/
# cp /etc/hosts /opt/mount1/
# ls -l /opt/mount1/
total 8
-rw-r--r--. 1 root root 146 Aug 27 22:17 fstab
-rw-r--r--. 1 root root 225 Aug 27 22:17 hosts

# ansible-playbook resize_lvm_filesystem_playbook1.yaml

```

- Verify the changed storage configuration. This is shown in Example 6-6.

Example 6-6 Verify the new storage configuration

```

# vgs
VG      #PV #LV #SN Attr   VSize  VFree
myvg    3   1   0 wz--n- <29.99g <28.99g

# lvs
LV      VG      Attr          LSize Pool Origin Data%  Meta%  Move Log Cpy%Sync
Convert
mylv1  myvg   -wi-ao---- 1.00g

# ls -l /opt/mount1/
total 8
-rw-r--r--. 1 root root 146 Aug 27 22:17 fstab
-rw-r--r--. 1 root root 225 Aug 27 22:17 hosts
# cat /opt/mount1/hosts
127.0.0.1      localhost localhost.localdomain localhost4
localhost4.localdomain4
::1           localhost localhost.localdomain localhost6
localhost6.localdomain6
192.168.159.133 bs-rbk-lnx-1.power-iaas.cloud.ibm.com bs-rbk-lnx-1

```

As we have verified in Example 6-6, the playbook has expanded the existing volume group, but the logical volume, file system and persistent mount point remains the same and the data in the file system is still accessible.

Resize existing file system

Now we will resize the existing file system using the following steps:

1. Create the playbook to resize the existing filesystem as shown in Example 6-7.

Example 6-7 Create resize_lvm_filesystem_playbook2.yaml playbook

```
# cat resize_lvm_filesystem_playbook2.yaml
---
- hosts: all
  vars:
    storage_pools:
      - name: myvg
        disks:
          - /dev/mapper/360050768108201d83800000000008e08p1
          - /dev/mapper/360050768108201d83800000000008e08p2
          - /dev/mapper/360050768108201d83800000000008e08p2
        volumes:
          - name: mylv1
            size: 10 GiB
            fs_type: xfs
            mount_point: /opt/mount1
  roles:
    - rhel-system-roles.storage
```

2. Now run the second playbook by running the following command:
ansible-playbook resize_lvm_filesystem_playbook1.yaml
3. Verify the storage configuration using the commands in Example 6-8.

Example 6-8 Verify the storage configuration

```
# vgs
VG #PV #LV #SN Attr VSize VFree
myvg 3 1 0 wz--n- <29.99g <28.99g

# lvs
LV VG Attr LSize Pool Origin Data% Meta% Move Log Cpy%Sync Convert
mylv1 myvg -wi-ao---- 10.00g

# df -h |grep '/opt/mount1'
/dev/mapper/myvg-mylv1 10G 106M 9.9G 2% /opt/mount1

# cat /opt/mount1/hosts
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
::1 localhost localhost.localdomain localhost6 localhost6.localdomain6
192.168.159.133 bs-rbk-lnx-1.power-iaas.cloud.ibm.combs-rbk-lnx-1
```

As we have verified in Example 6-8, the playbook has expanded the existing logical volume along with file system and the data is still accessible

Additional storage options can be done using Ansible playbooks. These can allow you to:

- Remove a file system
- Unmount a file system
- Remove LVM volume groups
- Remove logical volumes

For more information on how to use Ansible for these functions, refer to the following link:

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/automating_system_administration_by_using_rhel_system_roles/intro-to-rhel-system-roles_automating-system-administration-by-using-rhel-system-roles

in addition, the README file for the role will have more details on how to use the `rhel-system-roles.storage` role. It is located on your controller system at:

`/usr/share/ansible/roles/rhel-system-roles.storage/README.md`

6.2.2 Security and compliance

We have some selective security checklists from different industry standards like PCI-DSS, DoD and CIS, and consider sample baselines for this demonstration. Example 6-9 shows the `rhel-hardening-scanning` directory in our project directory that will have multiple sub-directories followed by a specific directory structure.

Example 6-9 List sub-directory of `rhel-hardening-scanning` project directory

```
# tree -d rhel-hardening-scanning/
rhel-hardening-scanning/
├── roles
├── rhel8hardening
│   ├── defaults
│   ├── files
│   │   └── pam.d
│   ├── handlers
│   ├── tasks
│   └── templates
└── rhel8scanning
    ├── defaults
    ├── files
    │   └── pam.d
    ├── tasks
    └── templates
```

Here is a basic introduction to these sub-directories and a description of their purpose:

► The `rhel-hardening-scanning` sub-directory

This is the project directory which has multiple sub-directories and contains the main playbooks that have been linked back to the role and/or other playbooks located in different sub-directories. For example:

- The file `rhel-hardening-scanning/playbook-rhel8hardening.yml` to harden the system according to our sample baselines.
- The file `rhel-hardening-scanning/playbook-rhel8scanning.yml` to scan the system according to our sample baselines and generate a report file.

► The `roles` sub-directory

This is a defined directory structure that allows you to develop reusable automation components by grouping and encapsulating related automation artifacts, like configuration files, templates, tasks, and handlers. We have omitted some directories the role does not use.

► The `rhel-hardening-scanning` sub-directory includes the following sub-directories:

defaults	Contains the default variables for the role and has defined all the required variables.
handlers	Will have a list of tasks that run only when a change is made on a machine, and run only after all the tasks in a particular play have been completed.
files	Contains all the files will be under this directory that the role deploys.
templates	Contains all the configuration template files will be under this directory that the role deploys.
tasks	Contains the list of tasks that the role executes and the main list of tasks will be in the file called main.yml.

Note: For more information on Ansible roles, see the [Playbook Guide](#). For assistance in developing a role, see [Developing an Ansible Role](#).

The list of files under the sub-directory of rhel-hardening-scanning project directory is shown in Example 6-10.

Example 6-10 Listing of files under the sub-directory of rhel-hardening-scanning project directory

```
# tree -f rhel-hardening-scanning/
rhel-hardening-scanning
├── rhel-hardening-scanning/ansible.cfg
├── rhel-hardening-scanning/hosts
├── rhel-hardening-scanning/playbook-rhel8hardening.yml
├── rhel-hardening-scanning/playbook-rhel8scanning.yml
├── rhel-hardening-scanning/roles
├── rhel-hardening-scanning/roles/rhel8hardening
│   ├── rhel-hardening-scanning/roles/rhel8hardening/defaults
│   │   └── rhel-hardening-scanning/roles/rhel8hardening/defaults/main.yml
│   ├── rhel-hardening-scanning/roles/rhel8hardening/files
│   │   ├── rhel-hardening-scanning/roles/rhel8hardening/files/chrony.conf
│   │   ├── rhel-hardening-scanning/roles/rhel8hardening/files/pam.d
│   │   │   ├── rhel-hardening-scanning/roles/rhel8hardening/files/pam.d/password-auth
│   │   │   ├── rhel-hardening-scanning/roles/rhel8hardening/files/pam.d/su
│   │   │   └── rhel-hardening-scanning/roles/rhel8hardening/files/pam.d/system-auth
│   │   └── rhel-hardening-scanning/roles/rhel8hardening/files/rsyslog.conf
│   ├── rhel-hardening-scanning/roles/rhel8hardening/handlers
│   │   └── rhel-hardening-scanning/roles/rhel8hardening/handlers/main.yml
│   ├── rhel-hardening-scanning/roles/rhel8hardening/tasks
│   │   ├── rhel-hardening-scanning/roles/rhel8hardening/tasks/main.yml
│   │   ├── rhel-hardening-scanning/roles/rhel8hardening/tasks/prerequisite.yml
│   │   ├── rhel-hardening-scanning/roles/rhel8hardening/tasks/section_A.yml
│   │   ├── rhel-hardening-scanning/roles/rhel8hardening/tasks/section_B.yml
│   │   ├── rhel-hardening-scanning/roles/rhel8hardening/tasks/section_C.yml
│   │   ├── rhel-hardening-scanning/roles/rhel8hardening/tasks/section_D.yml
│   │   ├── rhel-hardening-scanning/roles/rhel8hardening/tasks/section_E.yml
│   │   ├── rhel-hardening-scanning/roles/rhel8hardening/tasks/section_F.yml
│   │   └── rhel-hardening-scanning/roles/rhel8hardening/tasks/section_G.yml
│   └── rhel-hardening-scanning/roles/rhel8hardening/templates
│       └── rhel-hardening-scanning/roles/rhel8hardening/templates/login.defs.j2
└── rhel-hardening-scanning/roles/rhel8scanning
```

```

├── rhel-hardening-scanning/roles/rhel8scanning/defaults
│   └── rhel-hardening-scanning/roles/rhel8scanning/defaults/main.yml
├── rhel-hardening-scanning/roles/rhel8scanning/files
│   ├── rhel-hardening-scanning/roles/rhel8scanning/files/pam.d
│   │   ├── rhel-hardening-scanning/roles/rhel8scanning/files/pam.d/password-auth
│   │   ├── rhel-hardening-scanning/roles/rhel8scanning/files/pam.d/su
│   │   └── rhel-hardening-scanning/roles/rhel8scanning/files/pam.d/system-auth
│   └── rhel-hardening-scanning/roles/rhel8scanning/files/rsyslog.conf
├── rhel-hardening-scanning/roles/rhel8scanning/tasks
│   ├── rhel-hardening-scanning/roles/rhel8scanning/tasks/main.yml
│   ├── rhel-hardening-scanning/roles/rhel8scanning/tasks/postreport.yml
│   ├── rhel-hardening-scanning/roles/rhel8scanning/tasks/prerequisite.yml
│   ├── rhel-hardening-scanning/roles/rhel8scanning/tasks/section_A-report.yml
│   ├── rhel-hardening-scanning/roles/rhel8scanning/tasks/section_B-report.yml
│   ├── rhel-hardening-scanning/roles/rhel8scanning/tasks/section_C-report.yml
│   ├── rhel-hardening-scanning/roles/rhel8scanning/tasks/section_D-report.yml
│   ├── rhel-hardening-scanning/roles/rhel8scanning/tasks/section_E-report.yml
│   ├── rhel-hardening-scanning/roles/rhel8scanning/tasks/section_F-report.yml
│   └── rhel-hardening-scanning/roles/rhel8scanning/tasks/section_G-report.yml
└── rhel-hardening-scanning/roles/rhel8scanning/templates
    └── rhel-hardening-scanning/roles/rhel8scanning/templates/report.html.j2

```

Running rhel-hardening-scanning from ansible controller nodes

Target systems or managed nodes need some preparation to run the playbooks from the Ansible Controller Node. See “Getting started with Linux management” on page 31 for details on that process.

Example 6-11 shows how to run one of the provided playbooks to scan a system from the Ansible controller node using the Ansible command line.

Example 6-11 Run rhel-hardening-scanning from ansible controller node

```

# cd rhel-hardening-scanning/

# ls -l
total 32
-rw-r--r--. 1 root  root  19971 Aug 26 12:00 ansible.cfg
-rw-r--r--. 1 root  root1031 Aug 28 21:25 hosts
-rwxrwxrwx. 1 mhaque mhaque  123 Aug 26 11:43 playbook-rhel8hardening.yml
-rwxrwxrwx. 1 mhaque mhaque  125 Aug 26 11:58 playbook-rhel8scanning.yml
drwxrwxr-x. 4 mhaque mhaque    49 Aug 26 11:42 roles

# cat hosts
135.90.72.133

# ansible-playbook playbook-rhel8scanning.yml -u root

```

Run rhel-hardening-scanning from Ansible Automation Platform.

This example uses the Ansible Automation Platform to perform the scanning and hardening. The required resources – Credentials, Inventories, Projects and Job Templates, and Workflow Job Templates – have been already been created in the Ansible Automation Platform.

Figure 6-1 shows the screen for Job Templates and Workflow Job Templates configuration.

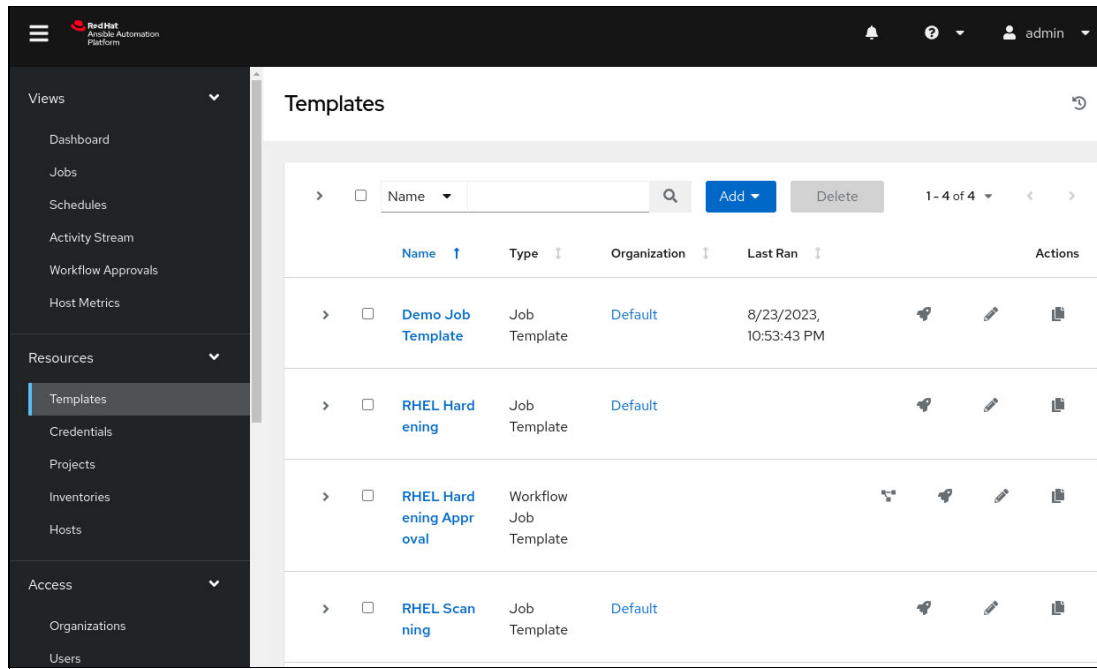


Figure 6-1 Job Templates and Workflow Job Templates.

One benefit of the Ansible Automation Platform – beyond the GUI interface provided – is the additional management functions provided such as the ability to require approval before starting any sensitive playbooks execution that may change system settings. Figure 6-2 shows defining a Workflow Job Template which is configured to require approval.

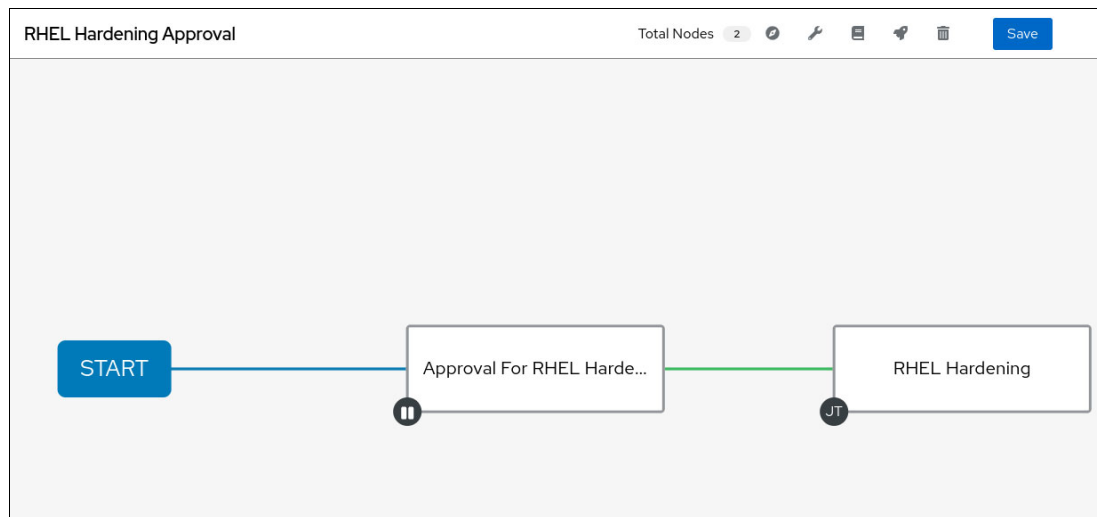


Figure 6-2 Workflow Job Templates configuration with approval.

Sample Scanning Report.

The scanning provides a very simple HTML template based report file that will generate in the managed nodes (target systems) or in the Ansible controller node. Figure 6-3 shows an example output from the top most part of the report file.

General Information			
Project Name		IBM RedBooks	
Computer Name		br-rh-lev-1-power-lan.cloud.ibm.com	
MAC Address		fa:8b:56:66:14:20	
IP Address		192.168.159.133	
Subnet Mask		255.255.255.248	
Default Gateway		192.168.159.129	
RHEL Version		8.6	
Total Tasks		35	
Non-compliant		16	
NA		0	
Execution Date		06-26-23_11:02	
Non-compliant Summary			
Section A1	Create separate partition for /tmp	FAILED	***Please Create Separate Partition Manually***
Section A2	Set noexec option for /tmp partition	FAILED	***Only Valid When /tmp is separated***
Section A3	Set nosuid option for /tmp partition	FAILED	***Only Valid When /tmp is separated***
Section A4	Set noexec option for /tmp partition	FAILED	***Only Valid When /tmp is separated***
Section A5	Create separate Partition for /var	FAILED	***Please Create Separate Partition Manually***
Section C2	Disable Send Packet Redirects	FAILED	
Section C3	Disable Source Routed Packet Acceptance	FAILED	
Section C4	Disable ICMP Redirect Acceptance	FAILED	
Section C5	Disable Secure ICMP Redirect Acceptance	FAILED	
Section D3	Create and Set Permissions on /tmp Log Files	FAILED	
Section D4	Configure rsyslog to Send Logs to a Remote Log Host	FAILED	Exception By Default
Section D5	Keep All Auditing Information	FAILED	
Section E3	Set User Group Owner and Permission	FAILED	
Section E4	Restrict /d Daemon	FAILED	
Section E5	Restrict access to Authorized Users	FAILED	
Section F4	Disable System Accounts	FAILED	
NA	Tasks	PASSED/FAILED	Remarks
Section A: Install Updates, Patches and Additional Security Software			

Figure 6-3 Top of the report files

Figure 6-4 shows an example of the bottom most part of the report file.

Section E5	Remove ND Client	PASSED	
Section C: Network Configuration and Firewall			
Section C1	Disable IP Forwarding	PASSED	
Section C2	Disable Send Packet Redirects	FAILED	
Section C3	Disable Source Routed Packet Acceptance	FAILED	
Section C4	Disable ICMP Redirect Acceptance	FAILED	
Section C5	Disable Secure ICMP Redirect Acceptance	FAILED	
Section D: Logging and Auditing			
Section D1	Install the rsyslog package	PASSED	
Section D2	Activate the rsyslog service	PASSED	
Section D3	Create and Set Permissions on /tmp Log Files	FAILED	
Section D4	Configure rsyslog to Send Logs to a Remote Log Host	FAILED	Exception By Default
Section D5	Keep All Auditing Information	FAILED	
Section E: System Access, Authentication and Authorization			
Section E1	Enable atacct Daemon	PASSED	
Section E2	Enable cron Daemon	PASSED	
Section E3	Set User/Group Owner and Permission	FAILED	
Section E4	Restrict /d Daemon	FAILED	
Section E5	Restrict access to Authorized Users	FAILED	
Section F: User Accounts and Environment			
Section F1	Set Password Expiration Days	PASSED	
Section F2	Set Password Change Minimum Number of Days	PASSED	
Section F3	Set Password Expiring Warning Days	PASSED	
Section F4	Disable System Accounts	FAILED	
Section F5	Set Default Group for root Account	PASSED	
Section G: Review User and Group Settings			
Section G1	Ensure Password Fields are Not Empty	PASSED	
Section G2	Verify No Legacy "*" Entries Exist in	PASSED	
Section G3	Verify No UID 0 Accounts Exist Other Than root	PASSED	
Section G4	Ensure root PATH Integrity	PASSED	
Section G5	Check Permissions on User Home Directories	PASSED	
RHEL Hardening Report generated by Red Hat Ansible Automation Tools.			

Figure 6-4 Bottom of the report files.

The sample playbooks for security and compliance for Linux on IBM Power System are freely available in the git repository for your use at:

<https://github.com/IBMRedbooks/SG248551-Using-Ansible-for-Automation-in-IBM-Power-Environments>

6.2.3 Fixes and Upgrades

One of the most important aspects of system security is keeping systems up to date with patches. You might have hundreds or thousands of Red Hat Enterprise Linux (RHEL) servers in their environments and keeping track of patches on servers can be challenging. But if critical patches are missed on systems, it could result in the systems being compromised, having unscheduled downtime, or other issues.

Red Hat Insights is a software-as-a-service (SaaS) offering that is included with your Red Hat Enterprise Linux subscription. It includes several capabilities to help with various aspects of management. The Patch capability can help customers understand which advisories are applicable in their environments, and can help automate the process of patching via Ansible playbooks.

For example, if a Red Hat Security Advisory were issued, you could go into the Insights Patch dashboard to see a list of systems in your environment that are impacted. With a few clicks from within the Patch dashboard, you can generate an Ansible Playbook that can automate the advisory installation. Figure 6-5 shows the GUI panel used to create the Ansible playbook.

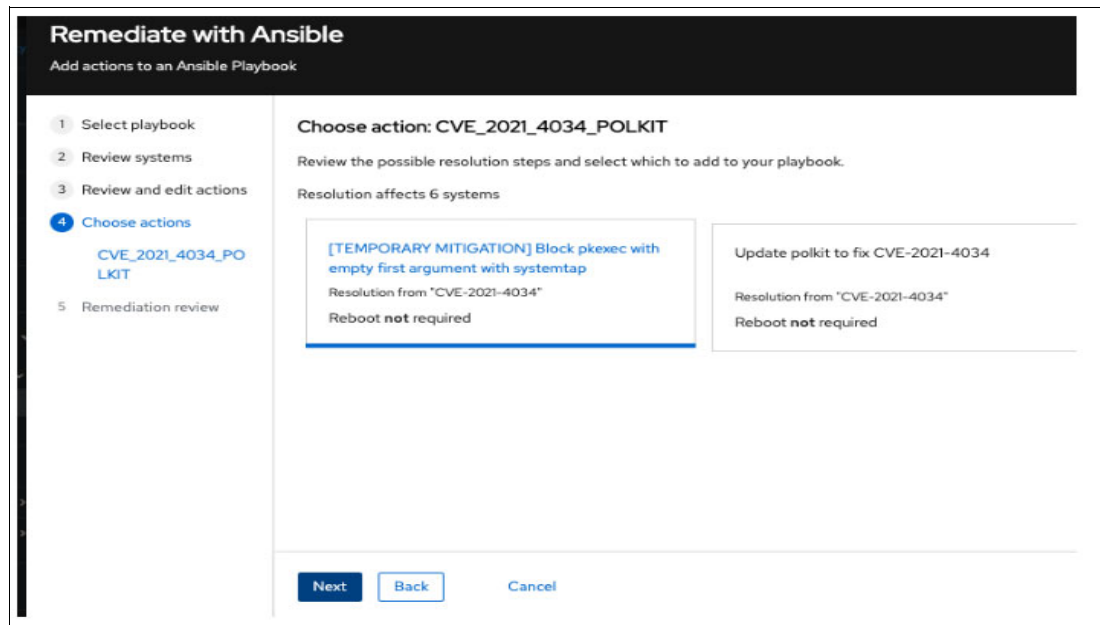


Figure 6-5 An example remediation playbook generation from Red Hat Insights¹

If you have Red Hat Smart Management, the Cloud Connector functionality lets you run the Ansible playbook right from the Insights web interface. Smart Management, Satellite and Cloud Connector are not required for use with Insights, and if you are in an environment without Red Hat Satellite you can still utilize Insights Patch and generate Ansible playbooks that can be downloaded and manually run.

For more information on getting started with the Red Hat Insights patch capability and how to download Ansible playbooks refer to this [Red Hat document](#).

¹ https://docs.redhat.com/en/documentation/red_hat_insights/1-latest/html/red_hat_insights_remediations_guide/creating-managing-playbooks_red-hat-insights-remediation-guide

Prerequisites

One of the following two options need to be in place to allow pulling patches and upgrades from the Red Hat repositories.

1. Use the Red Hat Satellite with the Red Hat Insights patch capability to enable and manage a standard operating environment for the patches or fixes repository. For more information see this [Red Hat patch management document](#).
2. Alternatively, you can provide an individual Red Hat Enterprise Linux subscription and connect with Red Hat Insights.

Example Ansible playbook for RHEL O/S patching.

We have a Power System RHEL LPAR (bs-rbk-ln x-1) that is already registered with Red Hat Subscription and connected with Red Hat Insights. Example 6-12 shows the command to validate that the LPAR is registered with Red Hat Insights.

Example 6-12 Command to verify insights registration of the system

```
[root@bs-rbk-lnx-1 ~]# insights-client --status
System is registered locally via .registered file. Registered at 2023-08-23T10:16:59.487964
Insights API confirms registration.
```

We have downloaded an Ansible playbook from the Red Hat Insights web console (<https://console.redhat.com/insights/inventory>) for advisory patches as seen in Figure 6-6.

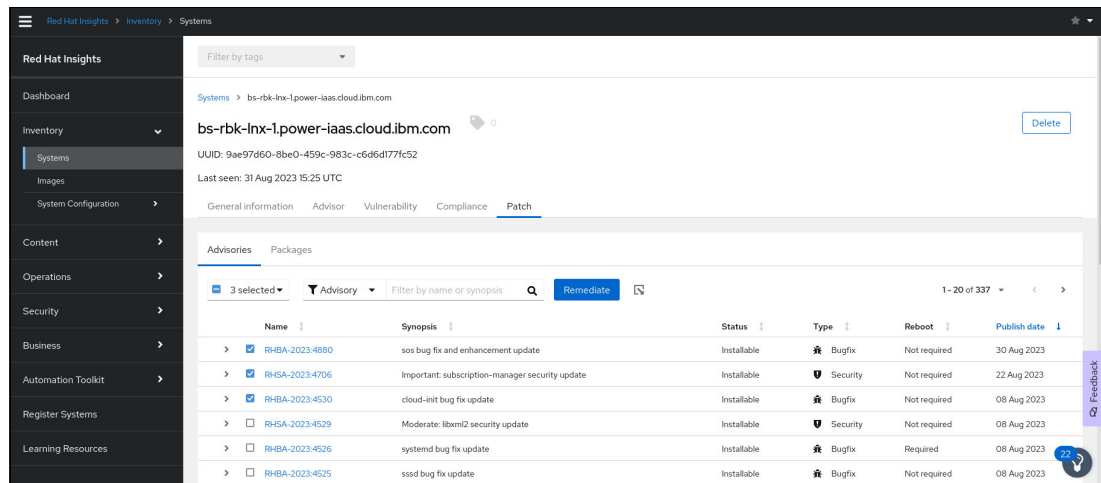


Figure 6-6 Creating a playbook (remediations) to apply patches from Red Hat Insights

After customizing some of the variables, the playbook shown in Example 6-13 is executed.

Example 6-13 Verify and run the O/S patch playbook

```
# cat os-patch-playbook-check.yml
---
# Upgrade the following packages:
# - Apply RHBA-2023:4530
# - Apply RHBA-2023:4880
# - Apply RHSA-2023:4706
- name: update packages
  hosts: "bs-rbk-lnx-1.power-iaas.cloud.ibm.com"
  vars:
```

```

        insights_issues: "--advisory RHBA-2023:4530 --advisory RHBA-2023:4880
--advisory RHSA-2023:4706"
requires_reboot: "true"
  become: true
  tasks:
- name: check for update
  shell: "{{ ansible_facts['pkg_mgr'] }}" check-update -q {{ insights_issues |
regex_search('--advisory ((FEDORA-EPEL-[\\w-]+)|(RH[SBE]A-20[\\d]{2}:[\\d]{4,5}))\\s*+')
}}"
  check_mode: no
  register: check_out
  failed_when: check_out.rc != 0 and check_out.rc != 100

- when: check_out.rc == 100
  name: upgrade package
  shell: "{{ ansible_facts['pkg_mgr'] }}" update -d 2 -y {{ insights_issues |
regex_search('--advisory ((FEDORA-EPEL-[\\w-]+)|(RH[SBE]A-20[\\d]{2}:[\\d]{4,5}))\\s*+')
}}"

- when: check_out.rc == 100
  name: set reboot fact
  set_fact:
    insights_needs_reboot: "{{requires_reboot}}"

# Reboots a system if any of the preceding plays sets the 'insights_needs_reboot' variable
to true.
# The variable can be overridden to suppress this behavior.
- name: Reboot system (if applicable)
  hosts: "bs-rbk-lnx-1.power-iaas.cloud.ibm.com"
  become: true
  gather_facts: false
  vars:
  tasks:
- when:
  - insights_needs_reboot is defined
  - insights_needs_reboot
  block:
  - name: Reboot system
    shell: sleep 2 && shutdown -r now "Ansible triggered reboot"
    async: 1
    poll: 0
    ignore_errors: true

  - name: Wait for system to boot up
    local_action:
      module: wait_for
      host: "{{ hostvars[inventory_hostname]['ansible_host'] |
default(hostvars[inventory_hostname]['ansible_ssh_host'], true) |
default(inventory_hostname, true) }}"
      port: "{{ hostvars[inventory_hostname]['ansible_port'] |
default(hostvars[inventory_hostname]['ansible_ssh_port'], true) | default('22', true) }}"
      delay: 15
      search_regex: OpenSSH
      timeout: 300
      become: false

# ansible-playbook os-patch-playbook-check.yml -u root

```

Note: Using customized yum/dnf commands will help create a customized list of rpms for RHEL patching when a RHEL minor version upgrade is not supported by a running application or is restricted by the application. Some example yum/dnf commands are shown here:

- yum --bugfix check-update
- yum --security check-update
- yum --advisory check-update
- yum --secseverity=Important check-update
- yum --sec-severity=Critical check-update
- yum check-update --cve CVE-2008-0947
- yum check-update --cve CVE-2008-0947
- yum check-update --bz 1305903
- yum updateinfo list
- yum updateinfo list security all
- yum updateinfo list bugfix all
- yum info-sec

For more information see the [YUM COMMAND CHEAT SHEET](#).

Figure 6-7 shows the results from executing the playbook.

```
[mhaque@munshi-lab ibm_redbooks]$ ansible-playbook os-patch-playbook-check.yml -u root
PLAY [update packages] *****
TASK [Gathering Facts] *****
The authenticity of host 'bs-rbk-lnx-1.power-iaas.cloud.ibm.com (135.90.72.133)' can't be established.
ECDSA key fingerprint is SHA256:d0C0DaaNGAzidwS+XZzU8Hu7cNwoJlRFEt5DQLb2es.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
ok: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]

TASK [check for update] *****
[WARNING]: Consider using the dnf module rather than running 'dnf'. If you need to use command because dnf 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: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]

TASK [upgrade package] *****
changed: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]

TASK [set reboot fact] *****
ok: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]

PLAY [Reboot system (if applicable)] *****
TASK [Reboot system] *****
changed: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]

TASK [Wait for system to boot up] *****
ok: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com -> localhost]

PLAY RECAP *****
bs-rbk-lnx-1.power-iaas.cloud.ibm.com : ok=6  changed=3  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
```

Figure 6-7 Output of the O/S patch playbook execution

The system has been patched with selective advisory rpms and was successfully rebooted as some of the rpms required a system reboot.

6.2.4 Configuration tuning

We discussed the RHEL System Roles in section 6.2, “Day 2 operations in Linux servers” on page 268. These roles that can help you configure system services in your system. For example you can configure the following functions:

- ▶ Secure Shell (SSH) Server & Client
- ▶ Firewall & SELinux
- ▶ Time synchronization
- ▶ Networking
- ▶ Postfix (mail transfer agent)
- ▶ High availability clustering
- ▶ Kernel settings

Dynamic configuration files templates

Ansible templates allow you to create files dynamically by interpolating variables or using logical expressions such as conditionals and loops. It's useful to define configuration files that adapt to different contexts without having to manage additional files.

The Ansible template engine uses the Jinja2² template language, a popular template language for the Python ecosystem. Jinja2 allows you to interpolate variables and expressions with regular text by using special characters such as `{` and `{%`. By doing this, you can keep most of the configuration file as regular text and inject logic only when necessary, making it easier to create, understand, and maintain template files.

Note: For more information on how to create dynamic configuration files using Ansible templates refer to <https://www.redhat.com/sysadmin/ansible-templates-configuration>.

Jinja2 templates are files that use variables to include static values and dynamic values. One powerful thing about a template is that you can have a basic data file but use variables to generate values dynamically based on the destination host. Ansible processes templates using Jinja2.

Example 6-14 shows how to create a file called `index.html.j2` for an apache web server.

Example 6-14 Sample index.html.j2 jinja2 templates file

```
# cat index.html.j2
Welcome to {{ ansible_hostname }}
-The ipv4 address is {{ ansible_default_ipv4['address'] }}
-The current memory usage is {{ ansible_memory_mb['real']['used'] }}mb out of {{
ansible_memory_mb['real']['total'] }}mb
-The {{ ansible_devices | first }} block device has the following partitions:
  -{{ ansible_devices['vdb']['partitions'] | join('\n -')}}

```

Now, Example 6-15 shows how to use the template created in Example 6-14 in an Ansible playbook.

Example 6-15 Using index.html.j2 jinja2 templates file in the playbooks

```
***** some output removed *****
# Push index Config Template
  - name: push index.html template
    template:
      src: index.html.j2
      dest: /var/www/html/index.html
***** some output removed *****

```

Note: For more information on managing Apache web servers using jinja2 templates and filters refer to: <https://www.redhat.com/sysadmin/manage-apache-jinja2-ansible>.

Example of Firewall configuration using RHEL System Roles

You can use Ansible to do system configuration using the RHEL System Role.

In Example 6-16 on page 284 we show how to create a playbook file (`~/opening-a-port.yml`) which will allow incoming HTTPS traffic to the local host.

² <https://jinja.palletsprojects.com/en/3.1.x/>

Example 6-16 firewall to allow incoming HTTPS traffic to the local host

```
# cat ~/opening-a-port.yml
---
- name: Configure firewalld
  hosts: managed-node-01.example.com
  tasks:
    - name: Allow incoming HTTPS traffic to the local host
  include_role:
    name: rhel-system-roles.firewall
vars:
  firewall:
    - port: 443/tcp
      service: http
      state: enabled
      runtime: true
      permanent: true
```

Example of SSHD configuration using RHEL System Roles

Example 6-17 shows creating a playbook file (*~/config-sshd.yml*) to configure the `sshd` service to only allow `root` login (with a password) from a specific subnet.

Example 6-17 Configuring `sshd` service in the local host

```
# cat ~/config-sshd.yml
---
- hosts: all
  tasks:
    - name: Configure sshd to prevent root and password login except from specific subnet
  include_role:
    name: rhel-system-roles.sshd
vars:
  sshd:
    # root login and password login is enabled only from a particular subnet
    PermitRootLogin: no
    PasswordAuthentication: no
    Match:
      - Condition: "Address 192.0.2.0/24"
        PermitRootLogin: yes
        PasswordAuthentication: yes
```

Example of kernel parameter settings using RHEL System Roles

This section shows how to create a playbook file (*~/kernel-settings.yml*) which configures some kernel settings to provide better performance of the system. The playbook is defined in Example 6-18.

Example 6-18 Configuring kernel settings in the local host

```
# cat ~/kernel-settings.yml
---
- hosts: testingservers
  name: "Configure kernel settings"
  roles:
    - rhel-system-roles.kernel_settings
  vars:
kernel_settings_sysctl:
  - name: fs.file-max
    value: 400000
  - name: kernel.threads-max
```

```

    value: 65536
kernel_settings_sysfs:
  - name: /sys/class/net/lo/mtu
    value: 65000
kernel_settings_transparent_hugepages: madvise

```

6.3 Day 2 operations in AIX environments

In this section we describe some of the most common tasks on your AIX systems you can automate with Ansible. Please bear in mind that the list of the tasks is not 100% complete. These are just examples how you can automate your environments. You can use them to feed your thoughts and should definitely amend them before using in your environment.

6.3.1 Storage

Storage management is a very common task, many administrators face on daily basis. You can easily automate storage management tasks with Ansible and give your playbooks to your operating team to win more time for important tasks.

Getting information about existing devices

Before you start working with storage devices, you must understand what you have on the system and how you can access the information.

When an Ansible playbooks starts its first task is usually to collect some information about the running system - gathering facts. The information Ansible has collected is available through the variable `ansible_facts`. This variable has some parts regarding storage configuration. First of all there is a list called `ansible_facts.devices` with the information about all devices on the system, including storage devices. You can also find a list of all mounted file systems in `ansible_facts.mounts`. Your volume groups are listed in `ansible_facts.vgs`.

One of the first problems almost every young administrator has on AIX is the absence of “df -h” (human readable output of all mounted filesystems). Let us solve the problem with Ansible. As we know Ansible stores the information about mounted filesystems in `ansible_facts.mounts`. We only need to print this information as shown in Example 6-19.

Example 6-19 Print information about mounted file systems in human readable format

```

---
- name: print information about mounted file systems in human readable format
  host: all
  gather_facts: true

  tasks:
  - name: print mounted filesystems
    ansible.builtin.debug:
      msg: "{{ item.device }} {{ item.size_total | ansible.builtin.human_readable }} {{
item.size_available | ansible.builtin.human_readable }} {{ item.mount }}"
      loop: "{{ ansible_facts.mounts | sort(attribute='mount') }}"
      loop_control:
        label: "{{ item.device }}"

```

Remember that Ansible can be run on several hosts in parallel. So you just get “distributed df -h” command.

When working with storage you want to understand how you get your disks from a Virtual I/O Server. You can find the information by analyzing `ansible_facts.devices` tree. If you find disks of type “Virtual SCSI Disk Drive”, you are using VSCSI. If you find disks of type “MPIO IBM 2145 FC Disk” or similar, you are using NPIV. Example 6-20 shows using Ansible facts to differentiate NPIV and VSCSI disks in a SAN attached IBM DS8000®.

Example 6-20 Find different types of disks on AIX

```

---
- name: find different types of disks on AIX
  hosts: all
  gather_facts: true

  tasks:
  - name: find VSCSI disks
    ansible.builtin.set_fact:
      vscsi_disks: "{{ ansible_facts.devices | dict2items | community.general.json_query(q)
  }}"
    vars:
      q: "[?value.type == 'Virtual SCSI Disk Drive'].{ name: key }"
  - name: find NPIV disks
    ansible.builtin.set_fact:
      npiv_disks: "{{ ansible_facts.devices | dict2items | community.general.json_query(q)
  }}"
    vars:
      q: "[?contains(value.type, 'IBM 2145')].{ name: key }"
  - name: VSCSI disks on the system
    ansible.builtin.debug:
      var: vscsi_disks
  - name: NPIV disks on the system
    ansible.builtin.debug:
      var: npiv_disks

```

One of the very common tasks in AIX system administration is to create a volume group on a new disk. How can you find out which disk is new? With Ansible it is easy. You start `cfgmgr` to find new disks and then re-collect the information about devices. The difference between two fact sets is your new disk. Example 6-21 demonstrates this capability,

Example 6-21 Finding new disks on AIX

```

---
- name: find new disk
  hosts: all
  gather_facts: true

  tasks:
  - name: find all existing hdisks
    ansible.builtin.set_fact:
      existing_disks: "{{ ansible_facts.devices | dict2items |
community.general.json_query(q) }}"
    vars:
      q: "[?starts_with(key, 'hdisk')].{ name: key }"
  - name: search for new disks
    ansible.builtin.command:
      cmd: cfgmgr
    changed_when: false
  - name: renew facts
    ansible.builtin.gather_facts:
      ignore_errors: true
  - name: get new list of disks

```

```

    ansible.builtin.set_fact:
      disks_after_cfgmgr: "{{ ansible_facts.devices | dict2items |
community.general.json_query(q) }}"
  vars:
    q: "[?starts_with(key, 'hdisk')].{ name: key }"
  - name: get new disks
    ansible.builtin.set_fact:
      new_disks: "{{ disks_after_cfgmgr | ansible.builtin.difference(existing_disks) }}"
  - name: print new disks
    ansible.builtin.debug:
      var: new_disks

```

The resulting output from running the playbook is shown in Example 6-22.

Example 6-22 Output from the playbook

```

# ansible-playbook -i localhost, -c local find-new-disk.yml

PLAY [find new disk]
*****
*****

TASK [Gathering Facts]
*****
*****
[WARNING]: Platform aix on host localhost is using the discovered Python interpreter at
/opt/freeware/bin/python3.9, but future installation of
another Python interpreter could change the meaning of that path. See
https://docs.ansible.com/ansible-
core/2.14/reference_appendices/interpreter_discovery.html for more information.
ok: [localhost]

TASK [find hdisks]
*****
*****
ok: [localhost]

TASK [search for new disks]
*****
*****
ok: [localhost]

TASK [renew facts]
*****
*****
ok: [localhost]

TASK [get new list of disks]
*****
*****
ok: [localhost]

TASK [get new disks]
*****
*****
ok: [localhost]

TASK [print new disks]
*****
*****

```

```
ok: [localhost] => {
  "new_disks": [
    {
      "name": "hdisk6"
    }
  ]
}
```

PLAY RECAP

```
*****
*****
```

```
localhost           : ok=7   changed=0   unreachable=0   failed=0   skipped=0
rescued=0   ignored=0
```

After you find the disk, you may want to set some attributes for it like reserve policy or hcheck_mode. This can be done using the `ibm.power_aid.devices` module as shown in Example 6-23.

Example 6-23 Setting device attributes for new disks

```
- name: set attributes for new disks
  ibm.power_aid.devices:
    device: "{{ item.name }}"
    attributes:
      reserve_policy: "no_reserve"
      hcheck_mode: "enabled"
    loop: "{{ new_disks }}"
    loop_control:
      label: "{{ item.name }}"
```

Working with volume groups

After you found the disks to work with, you may want to create a new volume group on it. The module to work with volume groups is called `ibm.power_aid.lvg`. This is demonstrated in Example 6-24.

Example 6-24 Create new volume group on AIX

```
---
- name: create volume group
  hosts: all
  gather_facts: false

  tasks:
  - name: create volume group
    ibm.power_aid.lvg:
      vg_name: datavg
      pvs: hdisk6
      vg_type: scalable
      pp_size: 256
      state: present
```

The created volume group is automatically activated and ready for further work like creating logical volumes or file systems.

If you use the a variable `new_disks` created in Example 6-22 on page 287 instead of inputting the name manually, you must build a string pointing to the variable as shown in Example 6-25 on page 289.

Example 6-25 Create a volume group using list of disks

```
- name: create volume group
  ibm.power_aix.lvg:
    vg_name: datavg
    pvs: "{{ new_disks | map(attribute='name') | join(' ') }}"
    vg_type: scalable
    pp_size: 256
    state: present
```

A similar method can be used to delete an unneeded volume group. You need only two attributes – the volume group’s name and state – this is shown in Example 6-26.

Example 6-26 Delete a volume group

```
- name: delete volume group
  ibm.power_aix.lvg:
    vg_name: datavg
    state: absent
```

The volume group will be deleted even it is open (varied on), but it can’t be deleted if there are allocations (logical partitions) in it. In this case you must collect LVM-related information, unmount all filesystems which are located on the volume group, remove all logical volumes and then delete the volume group as shown in Example 6-27.

Example 6-27 Delete a volume group with logical volumes in it

```
---
- name: delete volume group
  hosts: all
  gather_facts: false
  vars:
    vgroupname: datavg

  tasks:
  - name: gather LVM facts
    ibm.power_aix.lvm_facts:
  - name: "get logical volumes on {{ vgroupname }}"
    ansible.builtin.set_fact:
      lvols: "{{ ansible_facts.LVM.LVs | dict2items | community.general.json_query(q) }}"
    vars:
      q: "[?value.vg == '{{ vgroupname }}'].{ name: key, mount: value.mount_point }"
  - name: unmount all filesystems
    ibm.power_aix.mount:
      state: umount
      mount_over_dir: "{{ item.mount }}"
      force: true
    loop: "{{ lvols }}"
  - name: remove all logical volumes
    ibm.power_aix.lvol:
      lv: "{{ item.name }}"
      state: absent
    loop: "{{ lvols }}"
  - name: delete volume group
    ibm.power_aix.lvg:
      vg_name: "{{ vgroupname }}"
      state: absent
```

Another common task is when you want to expand a volume group by adding new disks to it or to shrink it by removing unneeded disks. It is the same procedure as for creating and deleting a volume group. Adding a disk is demonstrated in Example 6-28.

Example 6-28 Add disk to a volume group

```
- name: add disk to volume group
  ibm.power_aix.lvg:
    vg_name: datavg
    pvs: hdisk6
    state: present
```

Removing a disk is shown in Example 6-29.

Example 6-29 Remove disk from a volume group

```
- name: remove disk from volume group
  ibm.power_aix.lvg:
    vg_name: datavg
    pvs: hdisk6
    state: absent
```

Remember that the disk must be empty before removing it. If you need to move all logical volumes to another disk use `migratepv` command. Currently, there is not a special module or role to free up a disk. However, you can use `ansible.builtin.command` which passes any command to be run as if from the CLI. This is shown in Example 6-30.

Example 6-30 Free up a disk by moving logical partitions to another disk

```
- name: move all LPs to another volume
  ansible.builtin.command:
    cmd: "migratepv hdisk6 hdisk5"
```

Working with logical volumes

We already deleted logical volumes in the previous section but still didn't create any logical volumes using Ansible. Let us do it using the logic shown in Example 6-31.

Example 6-31 Create logical volume

```
- name: create logical volume
  ibm.power_aix.lvol:
    vg: datavg
    lv: lv01
    lv_type: jfs2
    size: 1G
    state: present
```

Using this same logic with a logical volume name specified, but with state specified as *absent*, the logical volume will be deleted. This is shown in Example 6-32.

Example 6-32 Delete logical volume

```
- name: delete logical volume
  ibm.power_aix.lvol:
    lv: lv01
    state: absent
```

Sometimes you need to change already existing logical volumes. For example, a common failure during filesystem expansion occurs if the logical volume was sized too small. This failure is shown in Example 6-33.

Example 6-33 Maximum allocation for logical volume is too small

```
# chfs -a size=+1G /lv01
0516-787 extendlv: Maximum allocation for logical volume lv01
is 128.
```

You can change the *maximum allocation* or any other value by using “extra_opts” attribute for *ibm.power_aix.lvol*. This can be seen in Example 6-34.

Example 6-34 Set maximum allocation for logical volume using extra_opts

```
- name: set maximum allocation for logical volume
  ibm.power_aix.lvol:
    vg: datavg
    lv: lv01
    size: 1G
    extra_opts: "-x 512"
    state: present
```

Note: In this case, you must specify volume group name and the size of the logical volume even if it doesn't change. To see which options you can use in extra_opts attribute please refer to [chlv command](#).

Working with filesystems

To work with filesystems you can use different modules depending on the tasks you want to automate. You can use *ibm.power_aix.filesystem*, *ibm.power_aix.mount* or *ansible.builtin.mount*.

To create a new filesystem when you have an existing logical volume, you can specify the logical volume name as shown in Example 6-35.

Example 6-35 Create filesystem on already existing logical volume

```
- name: create filesystem
  ibm.power_aix.filesystem:
    filesystem: /lv01
    device: lv01
    fs_type: jfs2
    auto_mount: true
    permissions: rw
    attributes: agblksize=4096,logname=INLINE
    state: present
```

If you don't have a prepared logical volume you must specify a volume group name to be used to create the new logical volume and include the size of the logical volume that will be created as shown in Example 6-36.

Example 6-36 Create filesystem on a new logical volume

```
- name: create filesystem
  ibm.power_aix.filesystem:
    filesystem: /lv02
    vg: datavg
    fs_type: jfs2
```

```

auto_mount: true
permissions: rw
attributes: agblksize=4096,logname=INLINE,size=1G
state: present

```

After you have created a filesystem, you should mount it as shown in Example 6-37.

Example 6-37 Mount filesystem

```

- name: mount filesystem
  ibm.power_aix.mount:
    mount_dir: /lv01
    state: mount

```

If you want to change the mount point of an existing filesystem, there is no special module for this. You can unmount the filesystem, but you can't change it using *ibm.power_aix.filesystem* module. To do this, you must unmount the filesystem, execute the command **chfs**, and then mount the filesystem on the new location.

Example 6-38 Change mount point of a filesystem

```

---
- name: change filesystem mount point
  hosts: all
  gather_facts: false

  tasks:
  - name: unmount filesystem
    ibm.power_aix.mount:
      state: umount
      mount_over_dir: /old_mount
      force: true
  - name: change mount point
    ansible.builtin.command:
      cmd: chfs -m /new_mount /old_mount
  - name: mount filesystem
    ibm.power_aix.mount:
      state: mount
      mount_dir: /new_mount

```

Very often as an AIX administrator you need to change the size of a filesystem. One of the biggest advantages of AIX is that you can change your filesystem configuration dynamically. Example 6-39 shows how to expand an existing filesystem.

Example 6-39 Expand filesystem

```

- name: expand filesystem
  ibm.power_aix.filesystem:
    filesystem: /lv02
    state: present
    attributes: size+=1G

```

Example 6-40 shows how to shrink a filesystem.

Example 6-40 Shrink filesystem

```
- name: shrink filesystem
  ibm.power_aix.filesystem:
    filesystem: /lv02
    state: present
    attributes: size=-1G
```

Currently there is a known bug in the *ibm.power_aix.filesystem* module. If there was an error during **chfs** execution and no parameters of the original filesystem were changed, you will often still get a status returned status of *OK* instead of *FAILED*. One of the reasons why the **chfs** fails can be the maximum allocations value on the underlying logical volume as we saw “Working with logical volumes” on page 290. In this case we need to first find the new value for the maximum allocation and set it before changing the size of the filesystem as demonstrated in Example 6-41.

Example 6-41 Expand filesystem and change the underlying logical volume

```
---
- name: expand filesystem
  hosts: all
  gather_facts: false
  vars:
    fs: /lv02
    size: 10G

  tasks:
    - name: get LVM facts
      ibm.power_aix.lvm_facts:
    - name: find logical volume for the filesystem
      ansible.builtin.set_fact:
        lvol: "{{ ansible_facts.LVM.LVs | dict2items | community.general.json_query(q) |
first }}"
      vars:
        q: "[?value.mount_point == '{{ fs }}'].{ lvname: key, vgname: value.vg, mount:
value.mount_point, lps: value.LPs }"
    - name: find pp size for the logical volume
      ansible.builtin.set_fact:
        ppsize: "{{ ansible_facts.LVM.VGs | dict2items | community.general.json_query(q) |
first | human_to_bytes | int }}"
      vars:
        q: "[?key == '{{ lvol.vgname }}'].value.pp_size"
    - name: recalculate new size in bytes
      set_fact:
        newsize: "{{ size | human_to_bytes | int }}"
    - name: find new max lp alloc
      set_fact:
        maxlp: "{{ ((newsized | int) / (ppsize | int)) | round(0, 'ceil') | int }}"
    - name: set max lp to logical volume
      ibm.power_aix.lvol:
        vg: "{{ lvol.vgname }}"
        lv: "{{ lvol.lvname }}"
        size: "{{ lvol.lps }}"
        extra_opts: "-x {{ maxlp }}"
        state: present
    - name: expand filesystem
      ibm.power_aix.filesystem:
        filesystem: "{{ fs }}"
```

```
state: present
attributes: "size={{ size }}"
```

If you want to remove an existing filesystem, you should unmount it first as shown in Example 6-42.

Important: Please bear in mind that AIX automatically deletes the underlying logical volume with all data on it if you remove a filesystem.

Example 6-42 Remove filesystem

```
---
- name: delete filesystem
  hosts: all
  gather_facts: false

  tasks:
  - name: unmount filesystem
    ibm.power_aix.mount:
      mount_over_dir: /lv02
      state: umount
  - name: delete filesystem
    ibm.power_aix.filesystem:
      filesystem: /lv02
      state: absent
```

Since one of the common tasks in many some environments is to add new disks, create a new volume group on those disks and then create a new filesystem on it using 100% of the disk. This can be done by combining the code from the previous tasks to automate the whole workflow as shown in Example 6-43.

Example 6-43 Create a filesystem on a new disk using 100% of its space

```
---
- name: create new filesystem on a new disk
  hosts: all
  gather_facts: true
  vars:
    vgroupname: vgoral
    lvname: lvoral
    fsname: /oral

  tasks:
  - name: find hdisks
    ansible.builtin.set_fact:
      existing_disks: "{{ ansible_facts.devices | dict2items |
community.general.json_query(q) }}"
    vars:
      q: "[?starts_with(key, 'hdisk')].{ name: key }"
  - name: search for new disks
    ansible.builtin.command:
      cmd: cfmgr
      changed_when: false
  - name: renew facts
    ansible.builtin.gather_facts:
      ignore_errors: true
  - name: get new list of disks
    ansible.builtin.set_fact:
```

```

    disks_after_cfgmgr: "{{ ansible_facts.devices | dict2items |
community.general.json_query(q) }}"
  vars:
    q: "[?starts_with(key, 'hdisk')].{ name: key }"
  - name: get new disks
    ansible.builtin.set_fact:
      new_disks: "{{ disks_after_cfgmgr | ansible.builtin.difference(existing_disks) }}"
  - name: finish if no new disks are found
    ansible.builtin.meta: end_host
    when: new_disks | length == 0
  - name: set attributes for new disks
    ibm.power_aix.devices:
      device: "{{ item.name }}"
      attributes:
        reserve_policy: "no_reserve"
        hcheck_mode: "enabled"
    loop: "{{ new_disks }}"
    loop_control:
      label: "{{ item.name }}"
  - name: create volume group
    ibm.power_aix.lvg:
      vg_name: "{{ vgname }}"
      pvs: "{{ new_disks | map(attribute='name') | join(' ') }}"
      vg_type: scalable
      pp_size: 256
      state: present
  - name: refresh LVM facts
    ibm.power_aix.lvm_facts:
      component: vg
  - name: get size of the new volume group
    ansible.builtin.set_fact:
      vgsizes: "{{ ansible_facts.LVM.VGs | dict2items | community.general.json_query(q) |
first | int }}"
    vars:
      q: "[?key == '{{ vgname }}'].value.total_pps"
  - name: create logical volume
    ibm.power_aix.lvol:
      vg: "{{ vgname }}"
      lv: "{{ lvname }}"
      lv_type: jfs2
      size: "{{ vgsizes }}"
      state: present
  - name: create filesystem
    ibm.power_aix.filesystem:
      filesystem: "{{ fsname }}"
      device: "{{ lvname }}"
      fs_type: jfs2
      auto_mount: true
      permissions: rw
      attributes: agblksize=4096,logname=INLINE
      state: present
  - name: mount filesystem
    ibm.power_aix.mount:
      mount_dir: "{{ fsname }}"
      state: mount
  - name: set permissions on the mount point
    ansible.builtin.file:
      path: "{{ fsname }}"
      owner: root
      group: system

```

```
mode: 0755
state: directory
```

Now you can compare how much time it would take to execute all of these commands manually and how much time it takes to run Ansible playbook. Also consider that with changing the inventory files and variables, this playbook can be reused multiple times.

6.3.2 Security

Security is a really big topic with a lot of nuances. It is impossible to describe the whole set of different configuration options which can be set in AIX to make it secure. We will go through some of them. You will find some more information about fixes, updates and general configuration tuning in the sections 6.3.3, “Fixes” on page 301 and 6.4, “Day 2 operations in IBM i environments” on page 307.

Managing users and groups

The first line in any security defense is to create user accounts for each user, set password rules for them and lock and delete them if they are not needed anymore. Let’s start with creating users on AIX. You can do it using `ibm.power_aix.user` module and it is very easy to use. This is shown in Example 6-44.

Example 6-44 Create a new user on AIX

```
- name: create user
  ibm.power_aix.user:
    name: user01
    state: present
    password:
      "{ssh512}06$t/IwQ/bp8ygs5J3j$09w3kfyg/Zct2R8n63t7gYDnt1gi7z50CFa5wPxj.hfwEX4ALFUx8805n8MBA
M5G1Ew7X4E7KG1ceNrp5XFW.."
    attributes:
      id: 1001
      shell: /usr/bin/bash
      home: /home/user01
      gecos: My fellow AIX admin
      pgrp: staff
      groups: staff,system,security
      fsize: -1
```

All possible values for the attributes section can be found in this [chuser description](#). These are standard AIX attributes you usually use with `mkuser` command.

During user creation you can set the user’s password. But it must be encrypted before. It is copied one-to-one into `/etc/security/passwd`. If you want that the user changes the password after the first login, add the attribute `change_passwd_on_login` to the task.

Unfortunately `ibm.power_aix.user` module is one of the few non-idempotent modules in the `ibm.power_aix` collection. It means you should use different states if you want to create or to modify user. Not as with some other modules, you set state to present and Ansible will figure it out and create a user if it doesn’t exist or change the specified attributes if it exists.

Let’s create simple password reset task for an AIX user. We generate a new password for the user and set the flag that the user must change the password at the first login. At the end we print the new generated password. This is shown in Example 6-45 on page 297.

Example 6-45 Password reset

```

---
- name: password reset
  gather_facts: false
  hosts: all
  vars:
    username: user01

  tasks:
  - name: generate random password
    ansible.builtin.set_fact:
      newpw: "{{ lookup('ansible.builtin.password', '/dev/null', chars=['ascii_lowercase',
'ascii_uppercase', 'digits', '.,-:_' ], length=8) }}"
  - name: encrypt password
    ansible.builtin.shell:
      cmd: "echo \"{{smd5}}$(echo \"{{ newpw }}\" | openssl passwd -aixmd5 -stdin)\\""
    changed_when: false
    register: newpw_enc
  - name: password reset
    ibm.power_aix.user:
      name: "{{ username }}"
      state: modify
      password: "{{ newpw_enc.stdout }}"
      change_passwd_on_login: true
  - name: show the generated password
    ansible.builtin.debug:
      msg: "The new password for {{ username }} is {{ newpw }}"

```

If you don't need the user anymore you can set the state to *absent* and the user will be deleted as shown in Example 6-46.

Example 6-46 Delete user

```

- name: delete user
  ibm.power_aix.user:
    name: user01
    state: absent

```

In similar way you can create and delete groups - see Example 6-47 for creating a group.

Example 6-47 Create group

```

- name: create group
  ibm.power_aix.group:
    name: group01
    state: present
    group_attributes: id=1000,adms=user01

```

Example 6-48 shows deleting a group.

Example 6-48 Delete group

```

- name: delete group
  ibm.power_aix.group:
    name: group01
    state: absent

```

Another very usual action on AIX systems is to add and to remove members from different groups. This is also easy to implement with Ansible and `ibm.power_aix.group` module. Adding members to a group is shown in Example 6-49.

Example 6-49 Adding new members to AIX group

```

---
- name: add members to group
  gather_facts: false
  hosts: all
  vars:
    newmembers:
      - john
      - johann
      - ivan

  tasks:
    - name: add members to group
      ibm.power_aix.group:
        name: security
        state: modify
        user_list_action: add
        user_list_type: members
        users_list: "{{ newmembers }}"

```

Removing a user from a group is shown in Example 6-50.

Example 6-50 Remove users from group

```

---
- name: add members to group
  gather_facts: false
  hosts: all
  vars:
    rmmembers: ivan

  tasks:
    - name: add members to group
      ibm.power_aix.group:
        name: security
        state: modify
        user_list_action: remove
        user_list_type: members
        users_list: "{{ rmmembers }}"

```

Note: Please note the inconsistency in the naming. The attributes `user_list_action` and `user_list_type` are in singular (user without s at the end), but the attribute `users_list` is in plural users with s at the end).

Managing IP filter configuration

IP filter is not the most popular AIX tuning and configuration option among AIX administrators, but more and more security departments require server-based firewalls to be enabled. With Ansible it is very easy to configure IP filter on AIX.

If you need to configure IP filter without Ansible, you need to go through several smitty menus or learn the command and parameters to set the filters using the command line. With Ansible you define a variable with your filter configuration and use one task to activate it as shown in Example 6-51 on page 299.

Example 6-51 AIX IP filter configuration

```

---
- name: configure AIX IP filter
  hosts: all
  gather_facts: false
  vars:
    aix_filter:
      - { action: permit, direction: outbound, s_addr: 0.0.0.0, s_mask: 0.0.0.0, d_addr:
0.0.0.0, d_mask: 0.0.0.0, protocol: all, description: 'allow outbound connections' }
      - { action: permit, direction: inbound, s_addr: 0.0.0.0, s_mask: 0.0.0.0, d_addr:
0.0.0.0, d_mask: 0.0.0.0, protocol: icmp, interface: all, description: 'allow incoming
pings' }
      - { action: permit, direction: inbound, s_addr: 0.0.0.0, s_mask: 0.0.0.0, d_addr:
0.0.0.0, d_mask: 0.0.0.0, protocol: tcp, d_opr: eq, d_port: 22, description: 'ssh' }
      - { action: permit, direction: inbound, s_addr: 0.0.0.0, s_mask: 0.0.0.0, d_addr:
0.0.0.0, d_mask: 0.0.0.0, protocol: all, d_opr: eq, d_port: 657, description: 'rnc' }
      - { action: permit, direction: inbound, s_addr: 0.0.0.0, s_mask: 0.0.0.0, d_addr:
0.0.0.0, d_mask: 0.0.0.0, protocol: tcp, d_opr: eq, d_port: 16191, description: 'caa' }
      - { action: permit, direction: inbound, s_addr: 0.0.0.0, s_mask: 0.0.0.0, d_addr:
0.0.0.0, d_mask: 0.0.0.0, protocol: tcp, d_opr: eq, d_port: 6181, description: 'caa' }
      - { action: permit, direction: inbound, s_addr: 0.0.0.0, s_mask: 0.0.0.0, d_addr:
0.0.0.0, d_mask: 0.0.0.0, protocol: tcp, d_opr: eq, d_port: 42112, description: 'caa' }
      - { action: permit, direction: inbound, s_addr: 0.0.0.0, s_mask: 0.0.0.0, d_addr:
0.0.0.0, d_mask: 0.0.0.0, protocol: tcp, d_opr: eq, d_port: 3901, description: 'nimsh' }
      - { action: permit, direction: inbound, s_addr: 0.0.0.0, s_mask: 0.0.0.0, d_addr:
0.0.0.0, d_mask: 0.0.0.0, protocol: tcp, d_opr: eq, d_port: 3902, description: 'nimsh' }
      - { action: deny, direction: inbound, s_addr: 0.0.0.0, s_mask: 0.0.0.0, d_addr:
0.0.0.0, d_mask: 0.0.0.0, protocol: all, log: true, description: 'deny and log everything
else' }
  tasks:
    - name: activate ip filter
      ibm.power_aix.mkfilt:
        ipv4:
          log: yes
          default: deny
          rules: "{{ aix_filter }}"

```

If you need to add a new rule, you can add it into the list and run the playbook again. The rule will be added. If you want to disable IP filter, you remove all filters, set it to allow by default and then close ipsec devices. This is shown in Example 6-52.

Example 6-52 Disable IP filter on AIX

```

---
- name: disable AIX IP filter
  hosts: all
  gather_facts: false

  tasks:
    - name: Remove all user-defined and auto-generated filter rules
      ibm.power_aix.mkfilt:
        ipv4:
          default: permit
          force: yes
          rules:
            - action: remove
              id: all
    - name: stop IPSec devices
      ibm.power_aix.devices:

```

```

    device: "{{ item }}"
    state: defined
  with_items:
    - ipsec_v4
    - ipsec_v6

```

If you want to be sure that IP filter rules are loaded in the correct order, you may remove all rules before loading them again.

Important: Please be careful changing IP filter configuration. Any mistake in the rules can lead to lost network connectivity.

Managing security settings with aixpert

The aixpert tool is a Swiss-army knife if you want to harden your system. With this single command you can check security settings and apply them. Ansible helps to implement it in a playbook to make it a part of a bigger process like server provisioning or hardening.

Let's first implement highly secure configuration on AIX.

Example 6-53 Apply highly secure configuration

```

- name: apply high secure configuration to AIX
  ibm.power_aix.aixpert:
    mode: apply
    level: high

```

Understand that it is aixpert which is doing the work this time, not Ansible. That's why if you apply the configuration twice, Ansible will run aixpert twice. It will not change your configuration the second time and it may even be faster than the first time run. But it is aixpert which checks and applies security settings, not Ansible.

You can check every time if the configuration is still intact by using check mode as shown in Example 6-54.

Example 6-54 Check AIX security configuration

```

- name: check applied settings
  ibm.power_aix.aixpert:
    mode: check

```

In case the configuration was changed, you can re-apply it as shown in Example 6-55.

Example 6-55 Check and re-apply AIX security settings if they were changed

```

- name: check applied settings
  ibm.power_aix.aixpert:
    mode: check
    ignore_errors: true
    register: aixpert_check
- name: re-apply security settings
  ibm.power_aix.aixpert:
    mode: apply
    level: medium
    when: aixpert_check.rc == 1

```

If you want to restore your previous settings, you can do it. This is possible as aixpert saves the old configuration and you can “undo” all the changes it made as is shown in Example 6-56.

Example 6-56 Restore security settings to their non-secure variant

```
- name: restore security settings
  ibm.power_aix.aixpert:
    mode: undo
```

6.3.3 Fixes

The topic of fixes, especially emergency or interim fixes, is a point of disagreement for many AIX administrators. Many administrators live according to the rule “if it works, don’t break it”. Unfortunately what many administrators misunderstand is the first part of the rule - “if it works”.

If a security issue was found in some AIX component, then it doesn’t work as it was designed to anymore. You don’t “break” it, you fix it. With Ansible it is easy to fix these security issues. If your environment supports Live Update, it can even be done while your systems are online and does not require any downtime.

Working with interim fixes

In this section we will present some examples of how Ansible can be used to manage interim fixes in your AIX environment. Some of the examples require direct access to the Internet to work. We understand that most of the corporate environments do not have any access to the Internet or have it only through proxies.

If your systems do not have access to the Internet then you are not able to automatically check and install fixes and you need to provide a method of checking acquiring the appropriate fixes somewhere in your environment. You need to have at least one server where you can download fixes which can then be distributed to the other systems.

If you do have access to the Internet, but only through a proxy, you will need to configure your proxy settings. Often it is as easy as to export the *https_proxy* variable in your environment, but sometimes it requires more complex work depending on the specifics of your environment. Setting up proxy configurations can be automated with Ansible, but it is beyond the scope of this discussion which is why we built our examples assuming that you have access to the Internet.

Note: During our tests we saw sometimes messages like **KeyError: 'message'**. It seems that FLRT service occasionally fails and delivers answers which are not understood by Ansible. Please repeat the task and next time it runs without any problems.

Check for AIX fixes

First we need to check if there are fixes needed in our AIX installation. This step requires Internet access as discussed earlier. In addition, this also requires that you have installed *wget* to download the fixes. The *wget* tool is an open source utility for retrieving files using the HTTP or FTP protocols and is available in the [AIX Toolbox](#) and can be installed with **dnf**. The module will automatically install flrtvc.ksh script from IBM website.

Example 6-57 shows how to generate a report on which fixes are available for your system.

Example 6-57 Generate report about available security fixes for AIX

```
- name: generate report about available fixes for the system
  ibm.power_aix.flrtvc:
    apar: sec
    verbose: true
    check_only: true
    register: flrtvc_out
- name: print the report
  ansible.builtin.debug:
    msg: "{{ flrtvc_out.meta['0.report'] | join('\n') }}"
```

If you want to see the report in a better format, you should export `ANSIBLE_STDOUT_CALLBACK=debug` or set it in the command as you call `ansible-playbook` as shown in Example 6-58.

Example 6-58 Setting debug before you run ansible-playbook

```
# ANSIBLE_STDOUT_CALLBACK=debug ansible-playbook -i inventory fixes-report.yml
```

Installing fixes

To automatically install the fixes you use the same command you used to check for fixes, but you need to remove “`check_only`” as shown in Example 6-59.

Example 6-59 Install all required security fixes on AIX

```
- name: install security fixes
  ibm.power_aix.flrtvc:
    apar: sec
    verbose: true
```

Ansible will download all of the fixes into `/var/adm/ansible`. As long as you have enough space in `rootvg` this is not a problem as Ansible will automatically expand the filesystem if it requires more space. You can choose a different location for temporary files by setting the attribute `path` to another directory.

Applying fixes to multiple servers

We showed you how to install required fixes on a single server, but often you will have multiple similar systems that you want to manage as a group. First, you have installed and tested the fixes in a non-production environment and now you want to apply those fixes to your production servers. Our example, shown in assumes that the fixes to be installed on the target production servers are downloaded and available in `/var/adm/fixes` on our Ansible controller node.

Example 6-60 Copy interim fixes from Ansible controller node to remote AIX server and install them

```
---
- name: find, copy and install security fixes
  hosts: all
  gather_facts: false
  vars:
    local_fixes_dir: /var/adm/fixes
    remote_fixes_dir: /var/tmp/fixes

  tasks:
    - name: find all fixes
      ansible.builtin.set_fact:
```

```

    fixes_list: "{{ fixes_list | default([]) + [ (item | basename) ] }}"
with_fileglob:
  - "{{ local_fixes_dir }}/*.*epkg.Z"
- name: create temporary directory for fixes on the target
  ansible.builtin.file:
    path: "{{ remote_fixes_dir }}"
    owner: root
    group: system
    mode: 0700
    state: directory
- name: copy fixes to the target
  ansible.builtin.copy:
    src: "{{ local_fixes_dir }}/{{ item }}"
    dest: "{{ remote_fixes_dir }}/{{ item }}"
    owner: root
    group: system
    mode: 0600
  loop: "{{ fixes_list }}"
- name: install fixes
  ibm.power_aix.emgr:
    ifix_package: "{{ remote_fixes_dir }}/{{ item }}"
    action: install
    from_epkg: true
    extend_fs: true
    force: true
  loop: "{{ fixes_list }}"
- name: remove fixes from the target
  ansible.builtin.file:
    path: "{{ remote_fixes_dir }}/{{ item }}"
    state: absent
  loop: "{{ fixes_list }}"

```

Sometimes especially before performing AIX update you want to remove all fixes. If you needed many commands to execute or a separate script earlier, now you can do it with only two tasks in an Ansible playbook.

Example 6-61 Remove all interim fixes from AIX

```

---
- name: remove all installed interim fixes
  hosts: all
  gather_facts: false

  tasks:
  - name: find all installed fixes
    ibm.power_aix.emgr:
      action: list
      register: emgr
  - name: uninstall fix
    ibm.power_aix.emgr:
      action: remove
      ifix_label: "{{ item.LABEL }}"
    loop: "{{ emgr.ifix_details }}"
    loop_control:
      label: "{{ item.LABEL }}"

```

Installing service packs and updates from NIM lpp_source resources

Usual AIX environment has at least one NIM server. NIM server is a central focal point in the infrastructure, has the newest version of AIX and contains resources to manage NIM clients. NIM administrator creates resources which can be used by NIM clients (AIX servers) to install new software or to update existing.

Let's assume we have an AIX server with AIX 7.3 and want to update it to AIX 7.3 TL1 SP2. The server is registered as NIM client and has access to NIM resources. Example 6-62 shows an example of how you can update AIX using NIM's lpp_source resource. It checks that the AIX server is registered as NIM client validates that it does not already have the update.

Example 6-62 Update AIX server using NIM lpp_source from NIM server

```

---
- name: update AIX server using NIM
  gather_facts: false
  hosts: nim
  vars:
    client: aix73
    aixver: 7300-01-02-2320
    reboot: false

  tasks:
    - name: check if client is defined
      ansible.builtin.command:
        cmd: lsnim {{ client }}
      changed_when: false
      register: registered
      ignore_errors: true
    - name: stop if the client is not registered
      meta: end_play
      when: registered.rc != 0
    - name: get client version
      ansible.builtin.command:
        cmd: /usr/lpp/bos.sysmgmt/nim/methods/c_rsh {{ client }} '( LC_ALL=C
/usr/bin/oslevel -s) '
      changed_when: false
      register: oslevel
    - name: stop if the client is already updated
      meta: end_play
      when: oslevel.stdout == aixver
    - name: update client
      ansible.builtin.command:
        cmd: nim -o cust -a lpp_source={{ aixver }}-lpp_source -a fixes=update_all -a
accept_licenses=yes {{ client }}
    - name: reboot client
      ibm.power_aix.reboot:
        when: reboot

```

Of course you can do it in another direction - from NIM client as shown in Example 6-63.

Example 6-63 Update AIX using NIM's lpp_source from NIM client

```

---
- name: update AIX server using NIM
  gather_facts: false
  hosts: aix73
  vars:
    aixver: 7300-01-02-2320

```

```

reboot: false

tasks:
- name: check if NIM client is configured
  ansible.builtin.command:
    cmd: nimclient -l master
  changed_when: false
  register: registered
  ignore_errors: true
- name: stop if the client is not registered
  meta: end_play
  when: registered.rc != 0
- name: get client version
  ansible.builtin.command:
    cmd: oslevel -s
  changed_when: false
  register: oslevel
- name: stop if the client is already updated
  meta: end_play
  when: oslevel.stdout == aixver
- name: update client
  ansible.builtin.command:
    cmd: nimclient -o cust -a lpp_source={{ aixver }}-lpp_source -a fixes=update_all -a
accept_licenses=yes
- name: reboot client
  ibm.power_aix.reboot:
  when: reboot

```

In a similar way you can update any software which is packed in `lpp_source` on your NIM server.

6.3.4 Configuration tuning

Last what we would like to discuss is AIX configuration tuning. AIX is a modern UNIX operating system with a lot of traditions. There are a lot of places in AIX which you can tune. In the book we can take a look on several most popular ways of tuning AIX. We already looks into most of security settings in the chapter 6.3.2, “Security” on page 296.

Most of AIX settings are stored in so called stanza files. They have sections, attributes and values. You can change values of attributes by using `chsec` module as shown in Example 6-64.

Example 6-64 Change AIX settings in `/etc/security/login.cfg` and `/etc/secvars.cfg`

```

- name: set settings
  ibm.power_aix.chsec:
    path: "{{ item.file }}"
    stanza: "{{ item.section }}"
    attrs: "{{ item.attrs }}"
  loop:
    - { file: "/etc/security/login.cfg", section: "usw", attrs: { sulogfulldate:
"true", mkhomeatlogin: "true", pwd_algorithm: "sha512" } }
    - { file: "/etc/secvars.cfg", section: "groups", attrs: { domainlessgroups: "true"
} }

```

Of course this is not the only way to change AIX settings. You can use standard `ansible.builtin.template` and `ansible.builtin.copy` modules to set AIX settings in the configuration files.

Security is not the only reason to automate AIX configuration. Another reason can be maintaining performance baselines. An application vendor like Oracle or SAP can define some values you have to set up on AIX to achieve better performance. An AIX administrator usually does it by using tunables commands like `vmo`, `no`, `schedo` or by setting device attributes, all of which is possible with Ansible. Example 6-65 shows how we can set `reserve_policy` to `no_reserve` and `queue_depth` to 24 for each disk we find in the system.

Example 6-65 Setting hdisk attributes

```
---
- name: set hdisk attributes
  hosts: all
  gather_facts: true

  tasks:
    - name: find hdisks
      ansible.builtin.set_fact:
        disks: "{{ ansible_facts.devices | dict2items | community.general.json_query(q) }}"
      vars:
        q: "[?starts_with(key, 'hdisk')].key"
    - name: set hdisk attributes
      ibm.power_aix.devices:
        device: "{{ item }}"
        attributes:
          reserve_policy: no_reserve
          queue_depth: 24
        chtype: reset
      loop: "{{ disks }}"
```

Example 6-66 shows how we can set some popular network tunables using Ansible.

Example 6-66 Setting network tunables with Ansible

```
- name: set network options
  ibm.power_aix.tunables:
    action: modify
    component: no
    change_type: both
    tunable_params_with_value: "{{ no_tunables }}"
  vars:
    no_tunables: {
      tcp_recvspace: 262144,
      tcp_sendspace: 262144,
      udp_recvspace: 262144,
      udp_sendspace: 262144,
      rfc1323: 1,
      tcp_fastlo: 1,
      tcp_keepintvl: 60,
      ipforwarding: 0
    }
}
```

In similar way you can other tunables like `vmo`, `schedo`, `ioo`, `nfso`.

It is not possible to provide detailed descriptions of all of the options available nor can we describe each use case as every environment is different from. We have tried to provide a little guidance and an initial impression of what can be done on AIX using Ansible.

Remember, Ansible has a very vibrant ecosystem. Every month you see new features in Ansible collections and Ansible itself. If you can't find some feature or module please report it.

Create an issue on Github or a topic on the IBM community site. You will help yourself and others to better automate by doing so.

6.4 Day 2 operations in IBM i environments

In this segment, we navigate the landscape of IBM i management, guided by the lens of Ansible automation. This encompasses a thorough investigation of essential elements, spanning storage, security, fix management, and configuration tuning. We expose actionable solutions and industry best practices, arming administrators with the means to deftly choreograph IBM i environments for optimal performance and accuracy.

6.4.1 Storage

In the following section, we explore in the context of storage management focused for IBM i. This segment aims to explore and demystify key storage-related tasks and configurations, harnessing the power of Ansible automation. Utilizing Ansible's capabilities, we aim to streamline and enhance the storage management experience for IBM i users, offering efficient solutions to common challenges.

Specifically, this section traverse three integral facets of storage management:

1. **Create/Delete storage volume with `os_volume`:** We explore into the intricacies of using the `os_volume` module to facilitate the creation and deletion of storage volumes. This module proves to be a versatile tool for managing storage resources with precision, ensuring that the storage landscape aligns with the dynamic requirements of IBM i.
2. **Attach storage volume with `os_server_volume`:** The `os_server_volume` module takes center stage as we explore the art of attaching storage volumes to IBM i servers. Here, we display how Ansible simplifies and accelerates the provisioning of storage resources, enhancing the scalability and adaptability of IBM i environments.
3. **Configure volumes to IASP:** The notion of Independent Auxiliary Storage Pools (IASPs) comes to the forefront as we unravel the intricacies of configuring volumes within this context. By exploring this aspect, we aim to permit users with the tools to create a storage infrastructure.

Enhancing IBM i storage management with the `os_volume` module

In IBM i storage management, the `os_volume` module plays a key role, enabling the creation and removal of cinder block storage volumes. This tool empowers administrators and operators to provision resources according to evolving workload needs. Users can define whether to create or remove volumes, aligning storage operations with operational demands.

The `os_volume` module interacts seamlessly with designated clouds, improving operations by providing default authentication values. Users can specify target cloud environments, ensuring secure communication between Ansible and the cloud.

Notably, this module allows granular control over volume size (gigabytes), accommodating precise resource allocation for IBM i. Volume naming enhances organization, and the module supports volume type specification, tailoring resources for various workloads.

Incorporating the `os_volume` module into Ansible simplifies storage management, promoting efficient provisioning and configuration. This exemplifies the dynamic storage landscape required by IBM i environments. Refer to Example 6-67 on page 308 for a playbook showcasing volume creation.

Example 6-67 Sample playbook to create a new volume on IBM i

```

---
name: Create New Volume
os_volume:
  state: present
  cloud: '{{ cloud_name }}'
  size: 150
  display_name: "{{ volume_name }}"
  volume_type: '70866ebf-0db5-4b12-86ed-0e838d593458'
register: volume_info
...

```

Attaching storage volumes to IBM i VMs with `os_server_volume` module

The `os_server_volume` module assumes a crucial role, facilitating the attachment of storage volumes to compute hosts. This module refines the process of linking volumes to IBM i virtual machines.

With its core functionality, `os_server_volume` presents administrators with the option to define the desired state of the resource, whether it can be present or absent. By accommodating named clouds, this module permits precise targeting of cloud environments for the operation. Default authentication values simplify setup and bolster secure communication between Ansible and the cloud.

The module's efficacy lies in its capacity to associate volumes with specific IBM i virtual machines. By providing the name of the target virtual machine and the volume, administrators can quickly attach storage resources, enabling the VM to access the necessary data. This module exemplifies the synergy between Ansible's automation capabilities and the storage demands of IBM i. Example 6-68 presents a sample playbook to attach volume to IBM i VM as follows:

Example 6-68 Sample playbook to attach volume to IBM i VM

```

---
name: Attach Volume to IBM i VM
os_server_volume:
  state: present
  cloud: '{{cloud_name}}'
  server: "{{ vm }}"
  volume: "{{ volume_info.id }}"
...

```

Note: For the effective utilization of the `os_server_volume` and `os_volume` modules, a prerequisite is the presence of IBM PowerVC as the orchestrator for your IBM i virtual machines. This integration emphasizes the significance of IBM PowerVC in bolstering the flexibility and resilience of your IBM i infrastructure.

Configuring IASP volumes with Ansible for IBM i

In IBM i storage management, configuring volumes within an Independent Auxiliary Storage Pool (IASP) is relevant for environments that uses IBM PowerHA, among others. Ansible's dynamic capabilities enhance this process, integrating storage resources into the IASP structure.

The `os_iasp_volume` role plays a crucial role in this configuration, efficiently orchestrating volume integration. The playbook first checks the IASP's existence and creates it if needed, showcasing Ansible's adaptability.

IASPs offer distinct benefits, enabling isolated disk unit management. The playbook demonstrates Ansible's integration with `os_iasp_volume`, effortlessly configuring non-configured disks into the IASP. This integration optimizes storage alignment, bolstering overall efficiency.

The playbook highlights Ansible's excellence in configuring IASP volumes. Administrators can expertly manage storage, ensuring a resilient landscape meeting evolving IBM i demands. Example 6-69 presents a sample setting up IASP volumes.

Example 6-69 Sample playbook to setup IASP volumes

```
---
- name: Initialize IBM i VM
  hosts: new_vm
  roles:
    - role: vm_iasp_configure
  vars:
    iasp_info:
      - {'iasp_name': 'demoiasp1', 'iasp_capacity': 0.5}
      - {'iasp_name': 'demoiasp2', 'iasp_capacity': 1}
...

```

6.4.2 Security and compliance

Security in IBM i environments is a multifaceted aspect that plays a key role in maintaining the integrity and confidentiality of critical data and operations. At its core, the system offers five distinct security levels denoted by the QSECURITY system value, ranging from levels 10 to 50. These levels enable administrators to tailor security measures to their specific needs. IBM i provides a comprehensive set of system values that allows administrators to define system-wide security settings, while also facilitating customization to meet diverse requirements across IBM Power Systems.

Digital signing emerges as a critical practice for ensuring the authenticity and integrity of software objects. This becomes especially pertinent when objects traverse the Internet or reside on media that could be susceptible to unauthorized modifications. The use of digital signatures, managed through mechanisms like the Verify Object Restore (**QVFOBJRST**) system value and the Check Manager tool, aids in detecting any unauthorized alterations.

Single sign-on (SSO) amplifies user convenience by allowing access to multiple systems with a single set of credentials. IBM facilitates SSO through Network Authentication Service (NAS) and Enterprise Identity Mapping (EIM), both utilizing the Kerberos protocol for user authentication. User profiles serve as a versatile tool to enforce role-based access control and personalize user experiences within the system. Group profiles extend this concept by centralizing authority assignments for groups of users.

Resource security is implemented through the concept of authorities, governing the ability to access objects. The system offers finely grained authority definitions, including subsets such as `*ALL`, `*CHANGE`, `*USE`, and `*EXCLUDE`. This mechanism applies not only to files, programs, and libraries but also to any object within the system.

Encryption stands relevant for security, with IBM i enabling data encryption at the ASP and Database Column levels. However, encryption operations can be carefully managed to mitigate performance implications. Security audit journals provide a means to monitor security effectiveness, allowing selected security-related events to be logged for review.

In the context of *Ansible for IBM i*, the integration of security measures is accommodated through a range of purpose-built modules. These modules permit diverse requirements,

including security and compliance checks, enabling administrators to configure, verify, and optimize security settings. Ansible for IBM i offers a robust ecosystem that allows administrators to ensure security compliance by referencing the *CIS IBM i Benchmark* documentation and employing regularly updated security compliance playbooks. This dynamic framework contributes to the creation of secure IBM i environments that align with modern security paradigms.

Overview of security management use case

The content here is designed to serve the security management use case. The playbooks involved offer you readily available samples that can be utilized as-is or adapted to suit your specific requirements.

Currently, the focus of these playbooks centers around security compliance checks. These playbooks are initially presented as basic examples, with plans to expand their contents based on security compliance suggestions outlined in the CIS IBM i Benchmark documentation.

Note: For more detailed guidance on implementing security practices on IBM i systems using Ansible, explore the provided use cases and security management resources available at GitHub repository [IBM i Security Management](#)

To stay up-to-date, it is recommended to regularly review this directory under the 'devel' branch.

In this part you see the explanation of the playbooks that are involved in this use case as follows:

1. **main.yml:** This playbook serves as an entry point, orchestrating the execution of all other playbooks contained within this directory. Running this playbook will initiate the execution of the entire suite.
2. **manage_system_values.yml:** The purpose of this playbook is to verify security-related system values against recommendations from the CIS IBM i Benchmark documentation. This playbook offers two separate YAML files for checking and remediating, along with three distinct modes of operation.
 - a. **system_value_check.yml:** Conducts a compliance check on system values by comparing them with the expected values.
 - b. **system_value_remediation.yml:** Provides remediation options based on user input, allowing remediation to be performed after a comprehensive review of the report.
3. **manage_user_profiles.yml:** This playbook leverages the 'ibmi_user_compliance_check' module and the 'ibmi_sql_query' module to assess user profile settings.
 - a. **user_profile_check.yml:** Performs a compliance check on user profiles.
 - b. **user_profile_remediation.yml:** Offers suggestions for remediation and carries out remediation actions based on user input.
4. **manage_network_settings.yml:** This playbook verifies a single network attribute setting by invoking the “Retrieve Network Attributes (RTVNETA)” command.
5. **manage_object_authorities.yml:** This playbook validates object authorities. It currently offers a basic example utilizing the 'ibmi_object_authority' module.

Additional Information

For a comprehensive understanding of how to execute a playbook dedicated to Secure Compliance for IBM i, it is recommended you refer Appendix A of the IBM Redbook publication *IBM Power Systems Cloud Security Guide: Protect IT Infrastructure In All Layers*,

REDP-5659. That section presents a detailed use case specifically focusing on security compliance for IBM Power Systems using Red Hat Ansible. Within the section titled “Security and Compliance with Red Hat Ansible for IBM i,” you will find a thorough walk through of the configuration process.

The section covers essential elements such as configuring the Ansible configuration file and the inventory file. It then proceeds to demonstrate the execution of the Ansible playbook. It is relevant to note that the playbook contains prompts related to specific checks for the managed node, remediation, or both. Additionally, prompts regarding the level of security definitions are also present, including Level 1 for Corporate and Enterprise Environment and Level 2 for High Security and Sensitive Data Environment.

During the execution of the playbook, you encounter fail messages within certain tasks. It is essential to interpret these messages as false-positive results. The playbook generates JSON files as final reports. These files serve as comprehensive reports to assess the outcomes on the managed IBM i node. The generated JSON files are stored in the `/tmp` directory.

In particular, the JSON report provides a systematic overview of the security management system values report. This detailed report sheds light on the security aspects analyzed during the execution of the playbook, offering insights into the compliance status of the IBM i environment.

6.4.3 Fix management

In Figure 6-8 we present an illustrative depiction of the process of IBM Fix Management, highlighting its key components and how they interact.

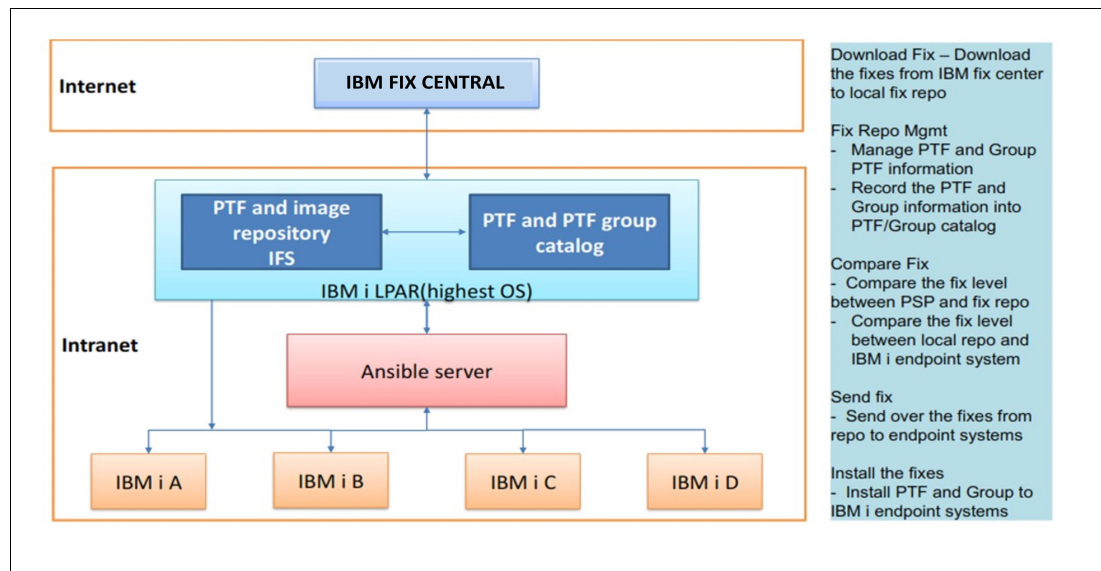


Figure 6-8 Advanced Fix Management workflow

The mechanism centers on IBM Fix Central, an online portal that establishes an Internet connection with multiple IBM i instances. These instances consist of two parts:

1. The first instance is the “PTF and Image Repository IFS.” This is where the acquired fixes are stored and where automatic detection of new PTF groups is enabled.
2. The second instance is the “PTF and PTF Group Catalog.” This stores information about downloaded fixes and assembles catalogs with new media.

To ensure coherence and robustness, a meticulous comparison of PTF versions against repository content and IBM i endpoints' systems takes place. This crucial coordination is orchestrated by the Ansible Controller, which effectively connects with each component within the intranet framework. Notably, the Ansible Controller is capable of installing PTFs and thoroughly examining the status of PTFs on the endpoints.

This dynamic use case, equipped with comprehensive functionalities, is available for download and adaptation. For more information, refer to GitHub repository: [Fix Management](#)

6.4.4 Configuration tuning

In this section, we explore into a set of playbooks that revolve around IBM i services. These playbooks, sourced from the GitHub repository at [IBM i Services](#), are designed to optimize various aspects of an IBM i environment. Let's take a closer look at each playbook and its purpose:

- ▶ `communications.yml`
- ▶ `message_handling.yml`
- ▶ `product_checking.yml`
- ▶ `system_health.yml`
- ▶ `work_management.yml`

Analyzing IBM i network communications

This playbook, accessible at [communications.yml](#), addresses the critical task of assessing and optimizing network communications within the IBM i environment. By utilizing various tasks, this playbook allows administrators to enhance the efficiency, security, and reliability of network connections.

The playbook commences with gathering essential facts, excluding default fact collection, through the `gather_facts: false` parameter. It uses the `ibm.power_ibmi` collection to execute its tasks smoothly. The `become_user_name` and `become_user_password` variables are employed for privilege escalation.

The playbook orchestrates a series of tasks that offer in-depth insights into network communications:

1. **Review most data transfer connections:** This task employs the `ibmi_sql_query` module to retrieve connections transferring substantial data (over 1 GB). The retrieved results are registered, providing crucial metrics about data flow. The subsequent debug task showcases these results, aiding administrators in evaluating resource utilization and potential bottlenecks.
2. **Analyze remote IP address detail for password failures:** Utilizing SQL queries, this task identifies and counts occurrences of failed password attempts from remote IP addresses within the past 24 hours. The gathered information is registered and presented through the debug task. This analysis assists administrators in detecting potential security threats and unauthorized access attempts.
3. **Review TCP/IP routes:** This task utilizes the `ibmi_sql_query` module to pinpoint TCP/IP routes with inactive local binding interfaces. By registering and displaying the details of these routes, administrators can identify and rectify potential network configuration issues that might impact communication reliability.

Optimizing message handling on IBM i

Discover the playbook at [message_handling.yml](#). This playbook, designed to improve message handling within the IBM i environment, allows administrators to proactively monitor and manage message queues and their responses. Facilitating smoother communication, the playbook helps maintain system health and performance.

Beginning with factual data collection disabled via `gather_facts: false`, the playbook harnesses the `ibm.power_ibmi` collection for execution.

The playbook orchestrates a series of tasks for thorough message handling optimization:

1. **Analyze next IPL status:** This task employs the `ibmi_sql_query` module to examine history log messages since the last Initial Program Load (IPL) to predict the nature of the next IPL. It assesses if it will be normal or abnormal based on specific messages. The results are registered, and an assert task ensures the next IPL is predicted to be normal, enhancing system predictability and stability.
2. **Examine system operator inquiry messages with replies:** This task employs SQL queries to retrieve system operator inquiry messages and their associated replies from the message queue 'QSYSOPR'. The gathered information is registered, offering administrators insights into system operator interactions and responses, thereby promoting efficient communication and issue resolution.
3. **Examine system operator inquiry messages without replies:** By analyzing system operator inquiry messages that have not received replies, this task enhances message handling efficiency. SQL queries extract relevant data from the 'QSYSOPR' message queue, and the results are registered and presented through the debug task.

License and product monitoring for IBM i

Explore the playbook at [product_checking.yml](#). This playbook provides administrators with valuable insights into the licensing and expiration status of products within the IBM i. It operates under the premise that ensuring the validity of licensed products is crucial for system health and compliance.

The playbook orchestrates two key tasks:

1. **Monitoring expiring licenses:** This task utilizes the `ibmi_sql_query` module to retrieve information about all licensed products and features set to expire within the next two weeks. This information is crucial for proactive license management, preventing disruptions due to expired licenses. Results are registered under the `expire_within_next_2_weeks` variable, providing administrators with actionable insights.
2. **Monitoring expiring licenses for installed products:** Similarly, this task employs SQL queries to retrieve details about licensed products and features that will expire within the next two weeks, specifically focusing on installed products. By considering only products marked as 'INSTALLED = YES,' administrators can prioritize active components requiring license renewal. Results are again registered under the `expire_within_next_2_weeks` variable for a comprehensive view.

IBM i system health analysis

Access the playbook at [system_health.yml](#), this playbook is designed to facilitate comprehensive system health analysis within the IBM i environment. Positioned to provide administrators with invaluable insights into system integrity and performance

The playbook orchestrates two central tasks, key for maintaining system health:

1. **Unofficial code inspection:** The playbook utilizes the `ibmi_sql_query` module to scrutinize the presence of any unofficial IBM i code within the QSYS library. By querying the QSYS2.OBJECT_STATISTICS view for objects labeled as '*PGM *SRVPGM,' the playbook identifies unsigned objects in the '*SYSTEM' domain. This inspection promotes security and stability. Results are registered under the `unofficial_code_check` variable.
2. **Large table identification:** Another task employs SQL queries to identify tables within the QSYS2.SYSLIMITS view that exceed a `CURRENT_VALUE` of 10,000,000. This highlights tables with a significant data volume, potentially indicating performance considerations. Results are captured in the `large_table` variable for further analysis.

IBM i work management analysis

Discover the playbook at [work_management.yml](#). This playbook focuses on IBM i work management analysis, delivering enhanced insights into job queue efficiency and system performance.

This playbook utilizes a series of critical tasks geared towards efficient work management:

1. **Scheduled job evaluation:** The playbook employs the `ibmi_sql_query` module to assess job schedule entries that are no longer effective due to explicit holding or scheduling limitations. The inspection, centered on the QSYS2.SCHEDULED_JOB_INFO view, targets 'HELD' and 'SAVED' status entries. Results, emphasizing maintained efficiency, are stored under the `job_schedule_status` variable.
2. **Job queue and temporary storage analysis:** The playbook capitalizes on SQL queries to uncover jobs awaiting execution within job queues (QSYS2.JOB_INFO). Furthermore, it scrutinizes the top four consumers of temporary storage based on memory pool usage, isolating jobs with temporary storage exceeding 1GB. These analyses contribute to optimized system resource allocation.
3. **Lock contention evaluation:** Through SQL queries, the playbook identifies jobs encountering excessive lock contention. By querying the QSYS2.ACTIVE_JOB_INFO view, it isolates jobs with combined database and non-database lock waits surpassing 2000. Insights are essential for maintaining operations.
4. **QTEMP resource utilization inspection:** The playbook evaluates host server jobs utilizing more than 10MB of QTEMP storage. With `qsys2.active_job_info`, jobs meeting this criterion are identified, enabling efficient resource allocation.



7

Chapter 7.

Future Trends and Directions

This chapter covers the future direction and plans of Ansible and IBM Power at a high-level. It also covers emerging trends in Ansible automation, and how we can maintain our Ansible code with products like Visual Studio Code and IBM watsonx Code Assistant for Red Hat Ansible Lightspeed.

The following topics are covered in this chapter:

- ▶ Ansible and IBM Power Systems Roadmap
- ▶ Roadmap for Ansible automation in the Power ecosystem

7.1 Ansible and IBM Power Systems Roadmap

In the past, modules specific to IBM Power had been created by the community. However, as you can see in section 1.5, “Ansible for Power” on page 29, IBM has been actively creating Ansible content for use across IBM Power servers since 2020. These collections are available on both Ansible Galaxy and Ansible Automation Hub.

IBM continues to develop new content, and improve existing content within these collections. If we take a look at the IBM Power AIX collections via Galaxy (for example), we can see the version release cycle in Figure 7-1.

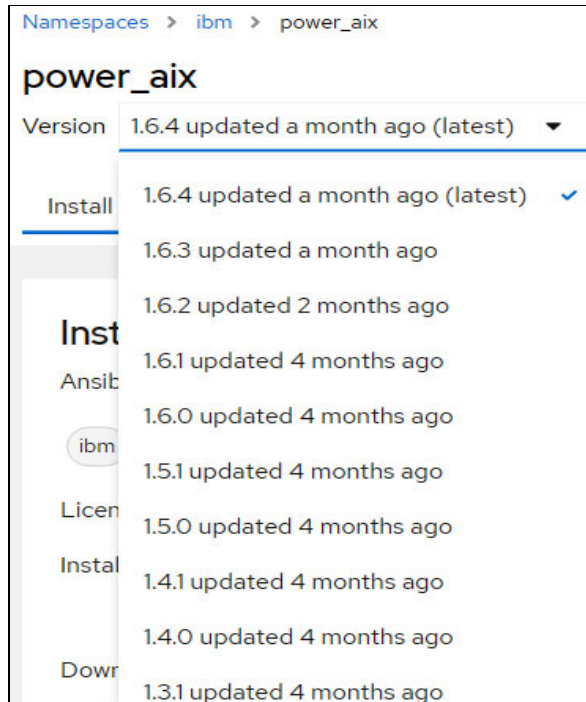


Figure 7-1 AIX collection release cycle

7.1.1 Working closely with the IBM Power collections and their contents

You can see what is included in the Ansible IBM Power collections and sample playbooks on the IBM github pages.

Viewing the content of the IBM Power collections

The content within each of the IBM Power collections can be viewed on IBM’s github pages, the link to each specific github repository can be accessed using the ‘repo’ link within Galaxy. Table 7-1 provides some of those links.

Table 7-1 GitHub content for IBM Power collections

Collection	Github URL
ibm.power_aix	https://github.com/IBM/ansible-power-aix
ibm.power_ibmi	https://github.com/IBM/ansible-for-i
ibm.power_hmc	https://github.com/IBM/ansible-power-hmc

Collection	Github URL
ibm.power_vios	https://github.com/IBM/ansible-power-vios

Within the github repositories you can see the code used to supply the collection, including the readme, the modules and some sample playbooks.

Raising an issue or suggesting enhancements to the IBM Power collection

As the IBM Power collections are available for anyone to see and use within Ansible Galaxy, we are able to request new features or suggest enhancements to the existing code. We can also raise issues with the code for the development team to review.

This is done within the 'issues' section of the github repository for the collection. You will see three options 'bug report', 'custom issue template' and 'feature request' as shown in Figure 7-2.

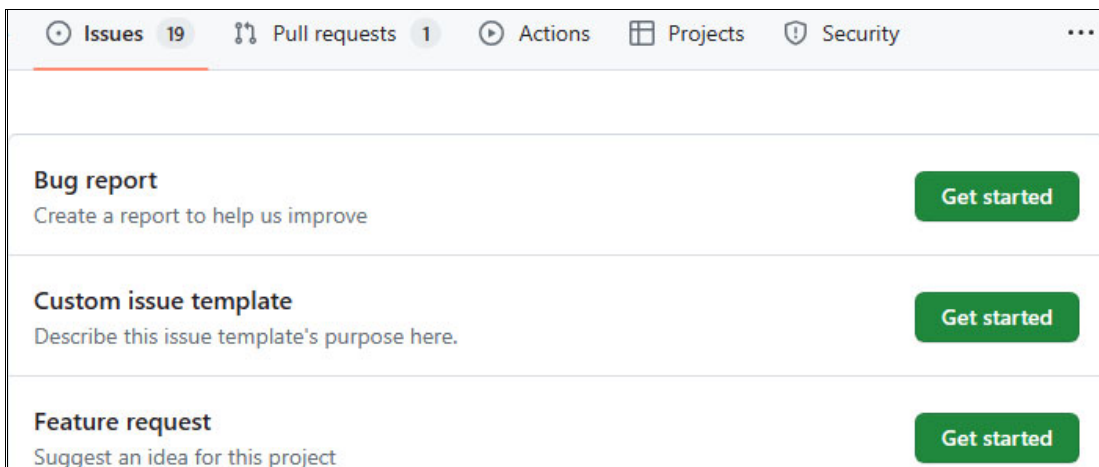


Figure 7-2 Raising a bug report or new feature request

You can also contribute to the collections by creating your own fork from the repository, making your changes to your fork and raising a pull request. This way the development team will see your proposed changes and either merge them into the collection or reject them.

7.2 Roadmap for Ansible automation in the Power ecosystem

IBM and Red Hat will continue to work on enhancements to the Ansible integrations of the IBM Power ecosystem. This includes updates to existing collections and implementations for provide additional functions involving machine learning and artificial intelligence.

7.2.1 Ansible Automation Platform on IBM Power

In June 2023, IBM and Red Hat announced that Ansible Automation Platform v2.4 was available as a Technical Preview on IBM Power. In December 2023, Red Hat announced the general availability of Ansible Automation Platform 2.4 on IBM Power (also announced was support for IBM Z and IBM LinuxOne). This means that as well as using Ansible to automate client endpoints on IBM Power (e.g. AIX, IBM i, VIO Server etc) you are able to run Ansible Automation Platform and all its components on IBM Power too.

What is new in Ansible Automation Platform v2.4 including the Technical Preview announcement can be seen in the link below.

<https://www.ansible.com/blog/whats-new-in-ansible-automation-platform-2.4>

Details on the general availability of Ansible Automation Platform can be found on this [Red Hat Blog](#).

Not only does this allow you to run the Ansible Controller (formerly Tower) on IBM Power, but all the other components that go to make up Ansible Automation Platform including execution environments, event-driven Ansible and automation hub.

7.2.2 Visual Studio Code

Visual Studio Code (also known as VS Code) is a very popular open source code editor available for Windows, Mac, Linux and also a web interface. Given the popularity of Visual Studio Code, it is the first GUI code editor to have a Ansible extension released by Red Hat.

The Ansible extension provides smart auto completion, syntax highlighting, validation, documentation reference, integration with *ansible-lint*, diagnostics, goto definition support, and command windows to run *ansible-playbook* and *ansible-navigator* tool for both local and execution-environment setups.

Visual Studio Code can be downloaded from <https://code.visualstudio.com/Download>.

Installing the Ansible extension

Installing the Ansible extension is done by opening VS Code, and clicking on the Extensions icon in the left panel and searching for the Ansible extension published by Red Hat, as shown in Figure 7-3.

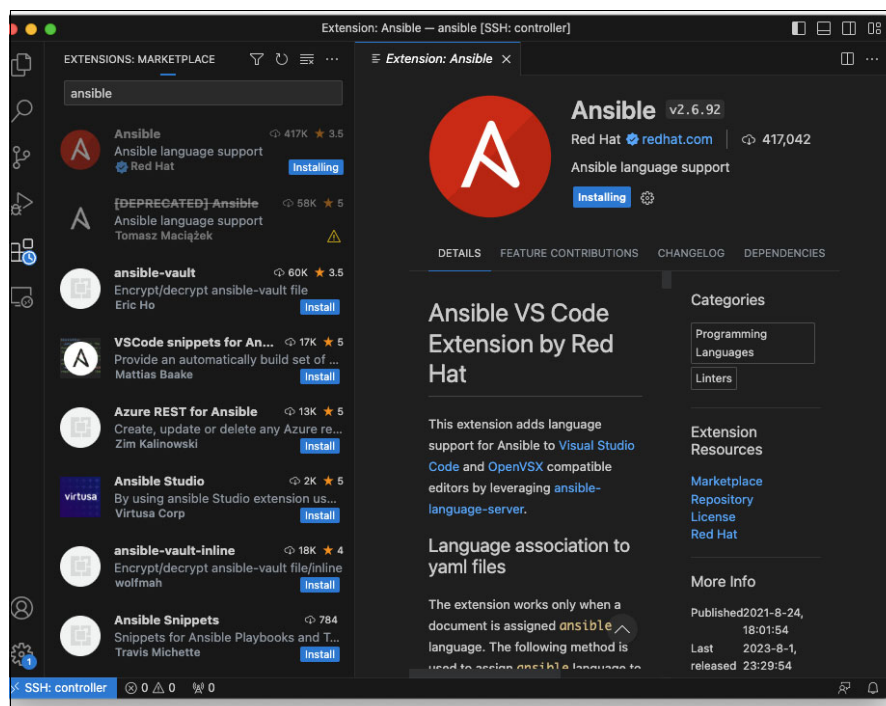


Figure 7-3 Installing the Ansible extension for Visual Studio Code

The next step is to open the folder that will contain your Ansible files using the Explorer icon in the top left as shown in Figure 7-4.

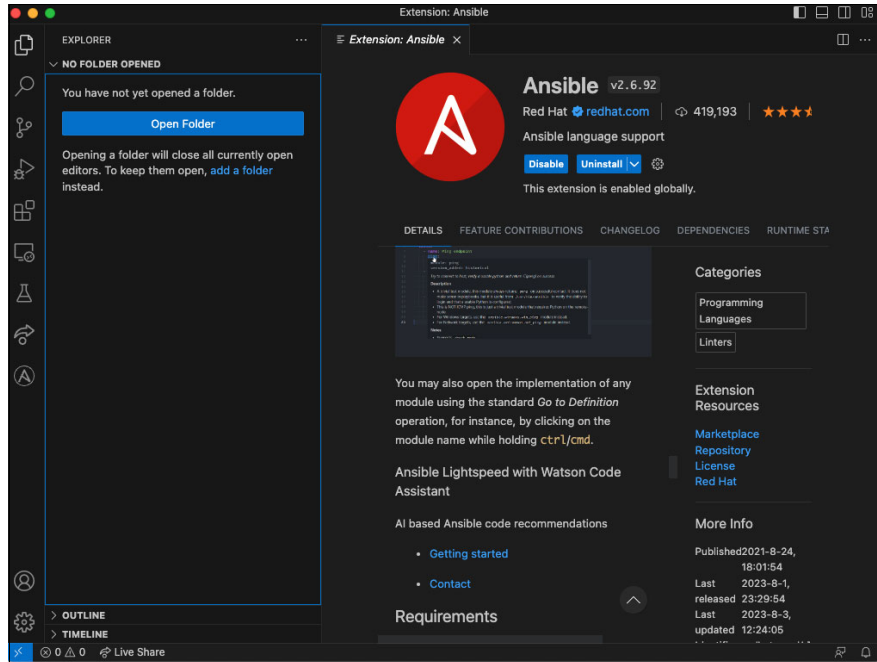


Figure 7-4 Open the folder containing your ansible files.

The first time you open a Ansible file (either .yaml or .yml), there are a few steps you will need to do.

1. Define which Python environment the Ansible extension should use, by clicking on the Python version indicator - which is located on the right hand of the Status Bar, as shown in Figure 7-5.

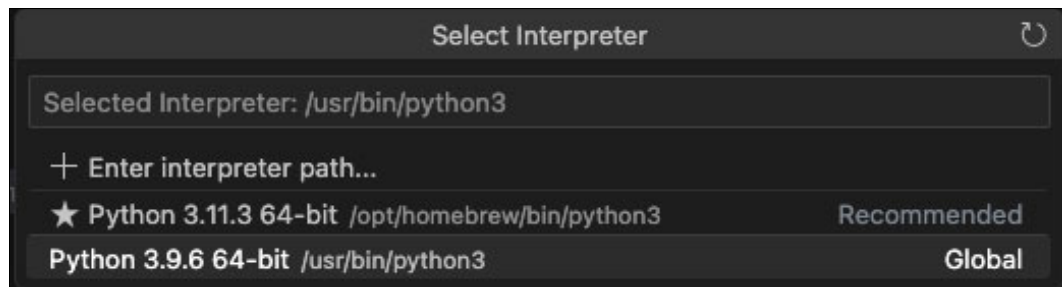


Figure 7-5 Select the desired Python environment

2. Associate the .yaml or .yml files with the Ansible file type, by clicking on the language indicator - which is located on the right hand of the Status Bar, and selecting Ansible from the drop down as shown in Figure 7-6 on page 320. The language indicator will probably be set to YAML before associating with the Ansible file type.

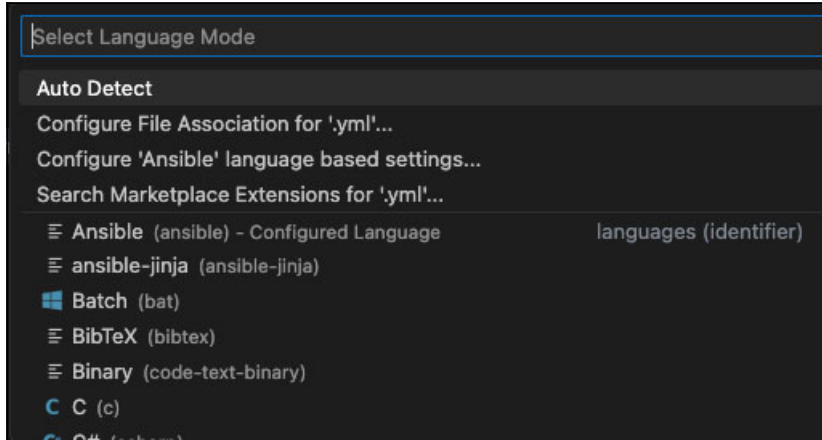


Figure 7-6 Associate YAML files with Ansible language

The Ansible extension should now recognize YAML files as Ansible language, offer syntax checking, documentation links, and other contextual aids to help you write Ansible code.

Note: If you have the *ansible-lint* package installed, this will automatically be integrated for syntax checking and code validation.

If you are working in a larger environment, you will probably not write, test and run your Ansible code on the same workstation where you installed Visual Studio Code. In this case, you will need to install the Remote SSH extension.

Installing the Remote SSH extension

Install the Remote SSH extension by clicking on the Extensions icon in the left navigation bar. Search for 'remote' and install the 'Remote - SSH' extension published by Microsoft, as shown in Figure 7-7.

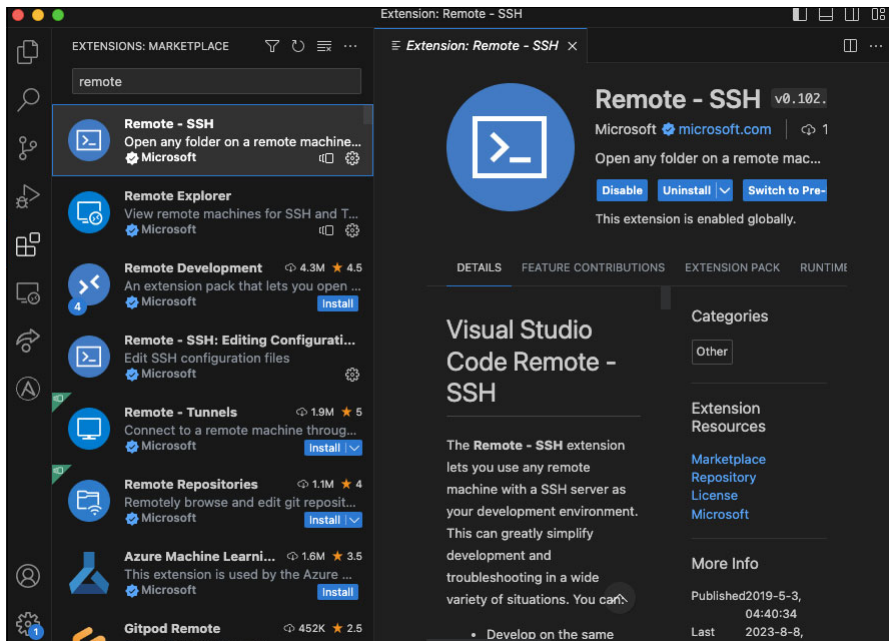


Figure 7-7 Installing Remote-SSH extension

Once installed you can open the VS Code Command Palette by pressing F1, and search for 'remote', as shown in Figure 7-8. Use either 'Remote-SSH: Connect to Host...' or 'Remote-SSH: Add New SSH Host' to enter the hostname and user credentials for the remote machine you will use to test and run your Ansible code.

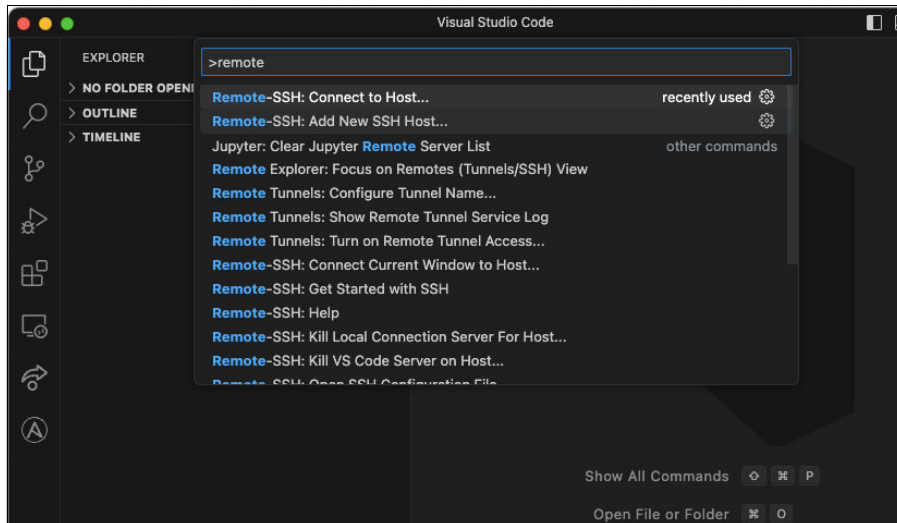


Figure 7-8 Connecting to a remote host with Remote-SSH extension

Contextual aids in Ansible extension

The Ansible extension for VS Code is designed to provide a number of contextual aids to writing and testing your Ansible code. Here are some of the features of the VS Code extension.

Syntax highlighting

Ansible module names, module option, and keywords are recognized and displayed in distinctive colors to allow the developer to see if the language syntax matches the intended purpose. Default colors will change depending on the color theme used. A sample in Visual Studio Dark theme is shown in Figure 7-9.

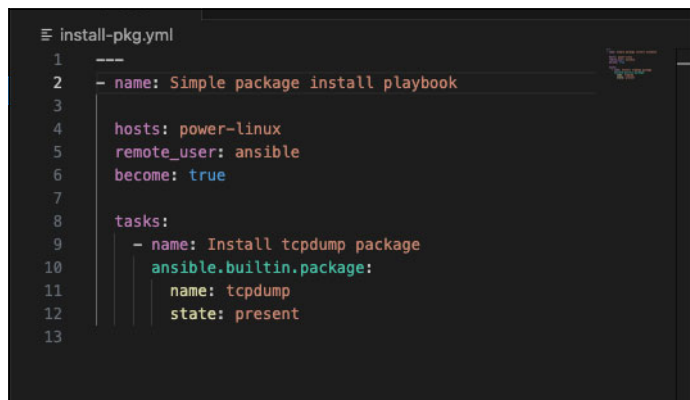


Figure 7-9 Ansible code syntax highlighting

Validation

The Ansible extension provides feedback regarding syntax as you type and any potential problems are shown in the 'Problems' tab of the integrated terminal, as shown in Figure 7-10 on page 322.


```

1  ---
2  - name: Simple package install playbook
3
4    hosts: power-linux
5    remote_user: ansible
6    become: true
7
8    tasks:
9      - name: Install tcpdump package
10         ansible.builtin.apt:
11           name: tcpdump
12           state
13

```

PROBLEMS 1 OUTPUT DEBUG CONSOLE ... Filter (e.g. text, **/*.ts...)

install-pkg.yml 1

- Implicit map keys need to be followed by map values... Ansible [YAML] [Ln 12, Col 9]

name: tcpdump
state

Figure 7-10 Code validation shown in Problems tab

Integration with ansible lint

When the `ansible-lint` package is installed it is integrated into the Ansible extension. `ansible-lint` is executed in the background whenever a file is opened or saved. Lines of code with errors are highlighted, and a more detailed description of the error is shown in the Problems tab as shown in Figure 7-11.

```

1  ---
2  - name: simple_package_install_playbook
3
4    hosts: power-linux
5    remote_user: ansible
6    become: true
7
8    tasks:
9      - name: install_tcpdump_package
10         ansible.builtin.package:
11           name: tcpdump
12           state: present
13

```

PROBLEMS 4 OUTPUT DEBUG CONSOLE ... Filter (e.g. text, **/*.ts...)

install-pkg.yml 4

- All names should start with an uppercase letter. `ansible-lint(name[casing])` [Ln 2, Col 1]
- All names should start with an uppercase letter. `ansible-lint(name[casing])` [Ln 9, Col 1]
- No new line character at the end of the file. `ansible-lint(yaml[new-line-at-end-of-file])` [Ln 13, Col 1]
- Trailing spaces `ansible-lint(yaml[trailing-spaces])` [Ln 13, Col 1]

Figure 7-11 Code syntax warnings generated by ansible-lint

Smart auto completion

As you type, the Ansible extension will offer suggestions to possible options depending on the context of the code, as shown in Figure 7-12. You can select and accept a suggestion, or disregard all options.

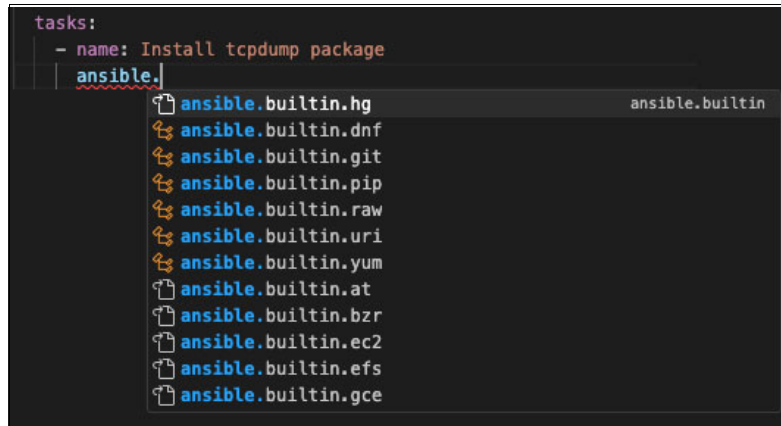


Figure 7-12 Auto completion of module name

Documentation reference

Hovering over a module name, module option, or keyword will show you a brief description of the item as a 'tooltip', as shown in Figure 7-13. You can display a full definition by either right clicking on the item, and selecting 'Go to definition' and the full definition will appear in a separate tab. Alternatively, you can select 'peek' from the menu to display the definition as a pop-up.

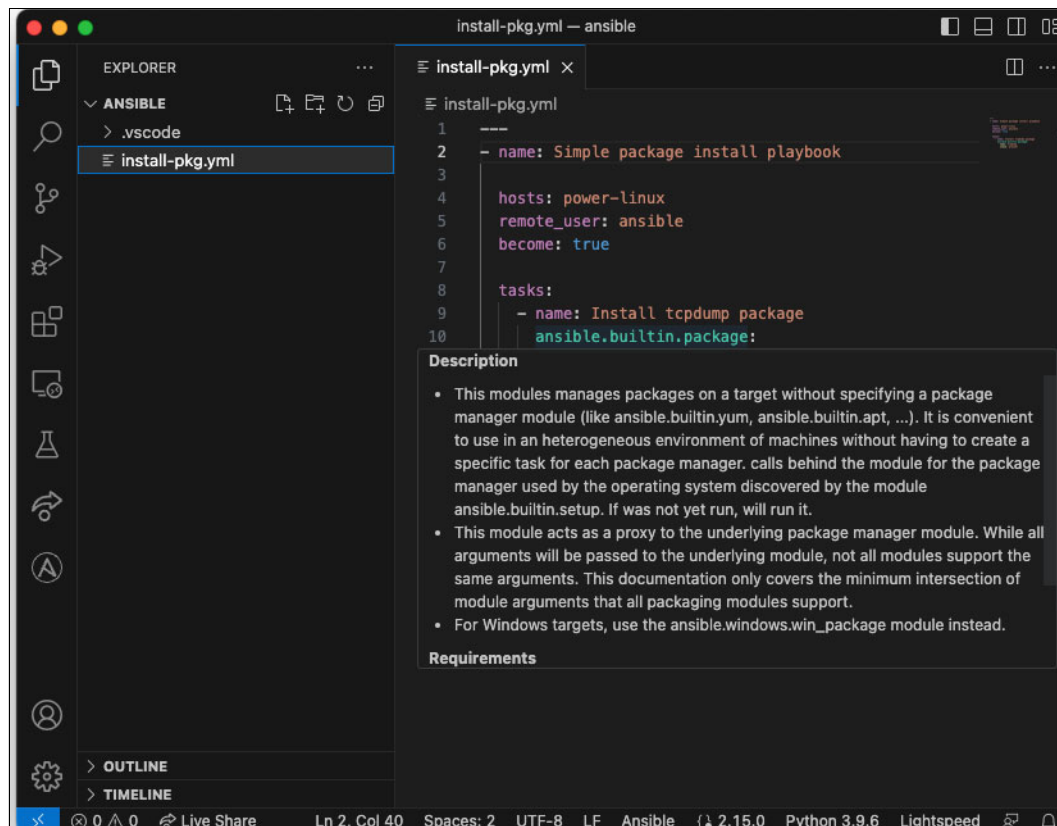


Figure 7-13 Module documentation show as a tooltip

Run playbook in integrated terminal

You can run a playbook from VS Code by right clicking on the playbook name in the Explorer tab, and selecting to run the playbook by either 'ansible-navigator' or 'ansible-playbook', as shown in Figure 7-14.

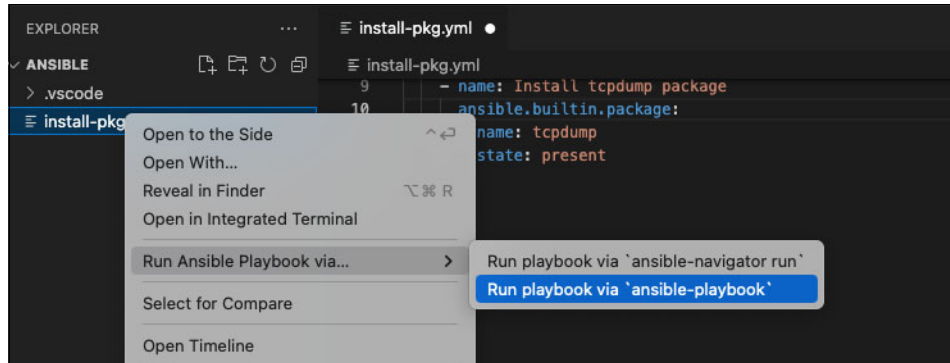


Figure 7-14 Run playbook in integrated terminal

Source control with git

With *git* installed on your workstation and/or your Ansible controller you can manage your source control using *git* from within Visual Studio Code.

The 'Source Control' icon on the left taskbar will show an overview of changed files that may need to be updated in your Github repository. Clicking on the icon will show the Source Control view that easily allows you to commit changes with a message and push to your repository as shown in Figure 7-15.

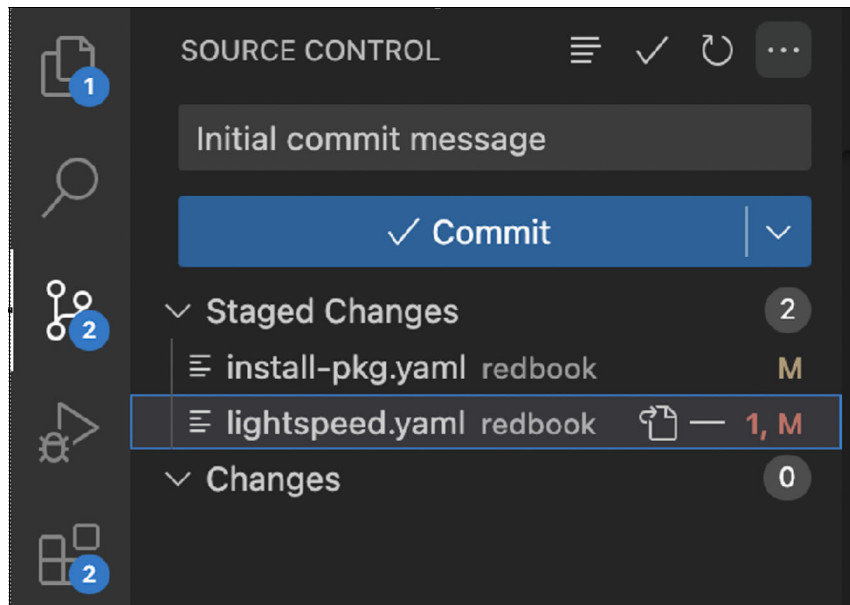


Figure 7-15 Files staged for commit to GitHub

Clicking on 'Views and More Actions' menu in the top right of the Source Control view will allow many more git operations such as clone, branch, and configure remote repo, as shown in Figure 7-16 on page 325.

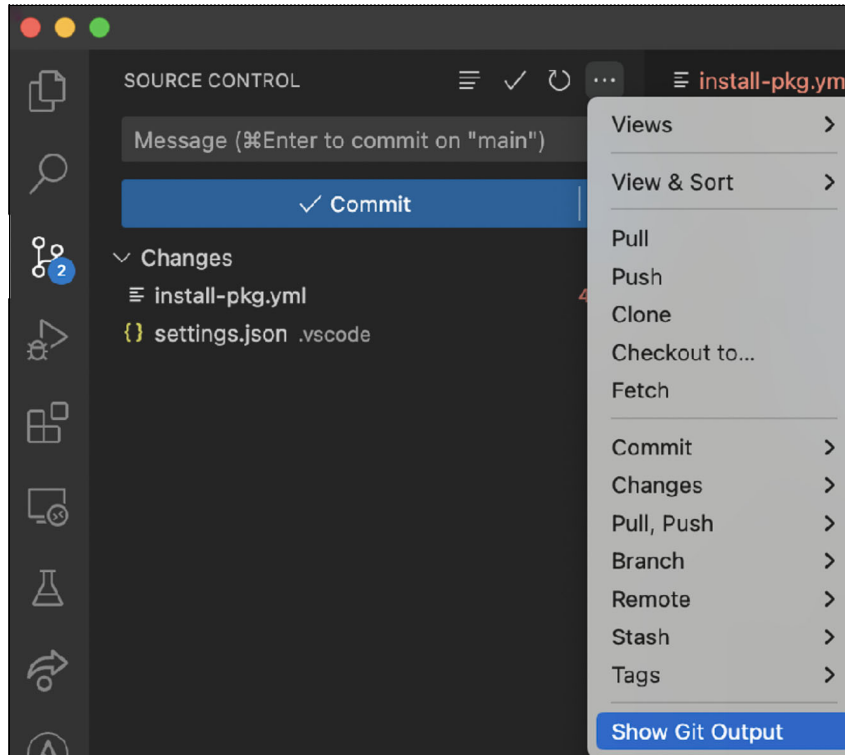


Figure 7-16 Git drop down menu

A full discussion of using git in Visual Studio is beyond the scope of this book, but more information can be found on various sources on the Internet including:

- ▶ <https://code.visualstudio.com/docs/sourcecontrol/overview>
- ▶ <https://www.ibm.com/garage/method/practices/code/visual-studio/>

7.2.3 IBM watsonx Code Assistant for Red Hat Ansible Lightspeed

IBM watsonx Code Assistant for Red Hat Ansible Lightspeed is a joint project between IBM and Red Hat that offers access to Ansible content recommendations through the use of natural language automation descriptions. This project is accessible through the integration of an IBM AI cloud service operated by Red Hat and the Ansible Virtual Studio Code plugin, and is offered to the Ansible community to use, without cost. This service uses, among other data, roles and collections that are available through the community website, Ansible Galaxy.

IBM watsonx Code Assistant for Red Hat Ansible Lightspeed has been released and is available for use for Red Hat customers. At this point IBM watsonx Code Assistant for Red Hat Ansible Lightspeed does not write complete playbooks, but can generate syntactically correct and contextually relevant content using natural language requests written in plain English text.

Getting Started

To enable IBM watsonx Code Assistant for Red Hat Ansible Lightspeed, you need the Visual Studio Ansible Extension from Red Hat discussed in 7.2.2, “Visual Studio Code” on page 318. You will also need a Red Hat login.

Once you have Visual Studio Code and the Ansible extension installed, go to the Settings panel for the Ansible extension as shown in Figure 7-17.

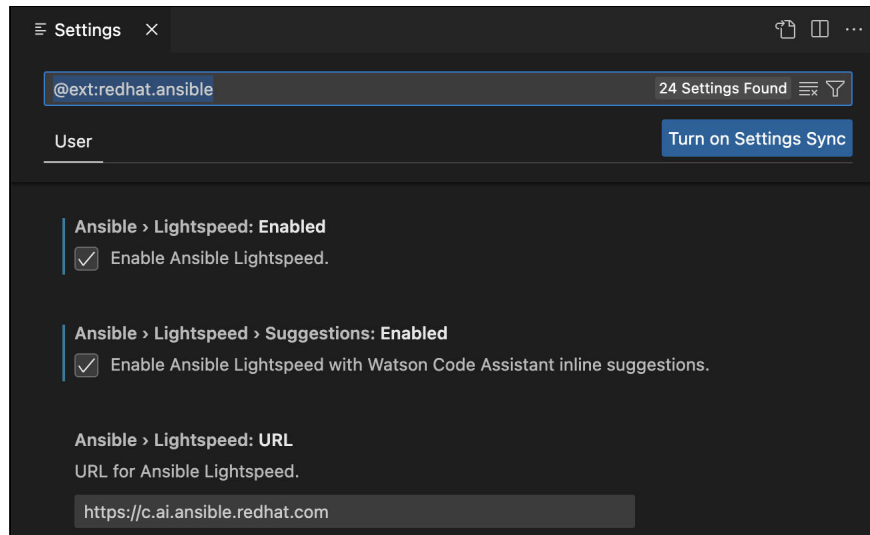


Figure 7-17 Enable Lightspeed in Ansible extension settings

Enable the following settings:

- ▶ Ansible > Lightspeed
- ▶ Ansible > Lightspeed > Suggestions
- ▶ Ansible > Lightspeed: URL should be `https://c.ai.ansible.redhat.com`

Then click on the Ansible icon (the letter A) in the left taskbar to display the Ansible Lightspeed Login panel as shown in Figure 7-18.

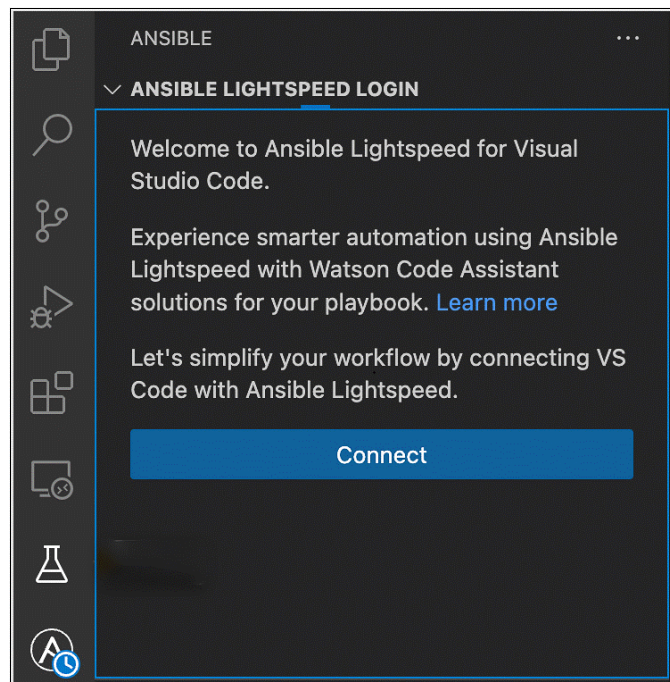


Figure 7-18 Connect to Ansible Lightspeed Login

Click on the Connect button in the 'Ansible Lightspeed Login' panel and you will be redirected to the IBM watsonx Code Assistant for Red Hat Ansible Lightspeed login web page. Follow the prompts to login with your Red Hat credentials as shown in Figure 7-19.

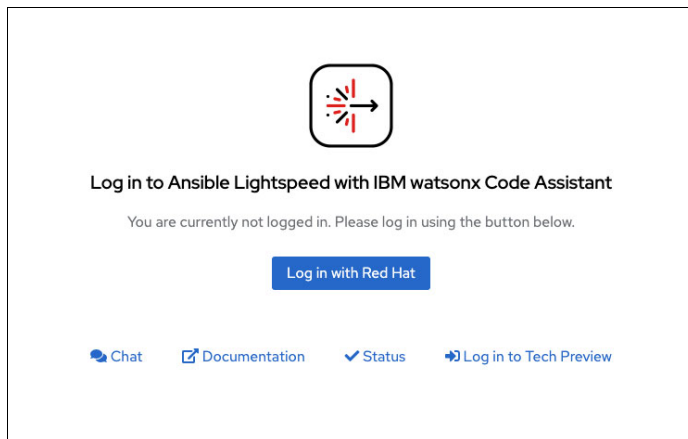


Figure 7-19 Lightspeed authentication screen

Once authenticated, accept the Terms & Conditions to enable IBM watsonx Code Assistant for Red Hat Ansible Lightspeed.

The final step is to authorize VS Code to interact with IBM watsonx Code Assistant for Red Hat Ansible Lightspeed extension by sending prompts and receiving code suggestions as shown in Figure 7-20. You should then see in the left taskbar that you are logged into IBM watsonx Code Assistant for Red Hat Ansible Lightspeed.

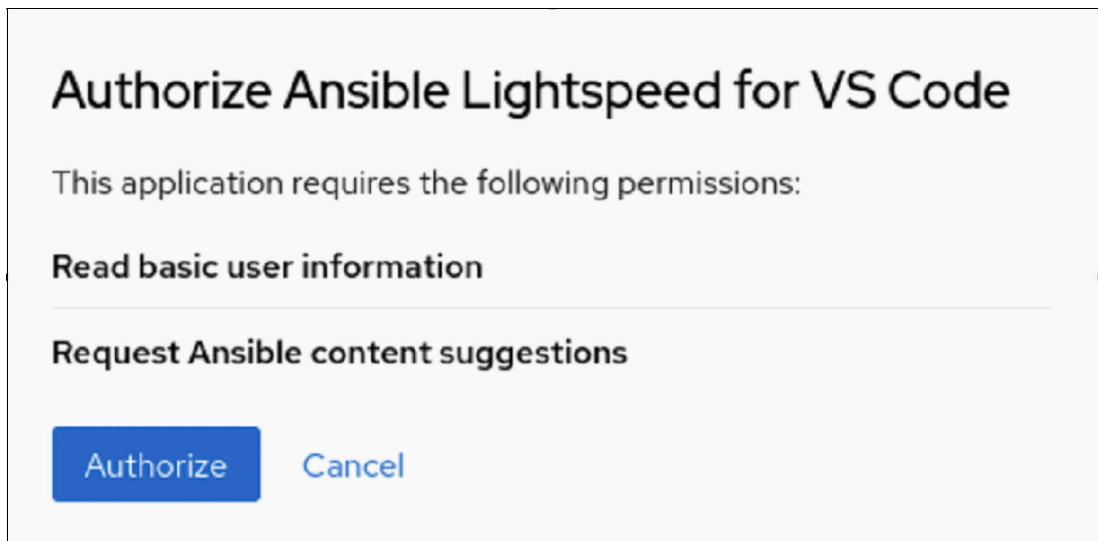


Figure 7-20 Authorize Ansible to interact with Ansible Lightspeed with Watson Code Assistant

Using IBM watsonx Code Assistant for Red Hat Ansible Lightspeed

To use IBM watsonx Code Assistant for Red Hat Ansible Lightspeed to get code recommendations for Ansible tasks, open a valid Ansible YAML file in the code editor. Check the bottom status bar of VS Code to ensure that the YAML file is recognized as Ansible language, and that Lightspeed is enabled.

Enter a task name and a description of what you want the task to do. Press **Enter** at the end of the line, and you should receive a code suggestion as shown in Figure 7-21 on page 328.


```

55 | - name: Copy logrotate.conf template to /etc/logrotate.conf
56 |   ansible.builtin.template:
      |     src: logrotate.conf.j2
      |     dest: /etc/logrotate.conf
      |     owner: root
      |     group: root
      |     mode: 420

```

Figure 7-21 Code suggestion from Lightspeed

The code suggestion will be shown in a gray font. Review the suggested code and either press **Tab** to accept the recommendation, or press **Esc** or **Enter** to dismiss it.

The source of the code suggestion is shown in the 'Ansible: Lightspeed Training Matches' tab of the panel below the code editor window as shown in Figure 7-22.

The screenshot shows a panel titled 'ANSIBLE: LIGHTSPEED TRAINING MAT' with several tabs: PROBLEMS (1), OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS (2), and ANSIBLE: LIGHTSPEED TRAINING MAT. The panel displays a tree view of search results:

- ▶ community.general.aix_lvg
- ▶ lefilament.install_servers
- ▼ ibm.power_aix.lvg
 - URL: https://galaxy.ansible.com/ibm/power_aix/lvg
 - Path:
 - Data Source: 7
 - License: gpl-3.0
 - Ansible type: 1
 - Score: 1.7595603

Figure 7-22 Source of code suggestion from Lightspeed

Note: The panel below the code editor shows various tabs including: Problems, Output, Debug Console and Integrated Terminal. It can be toggled on and off with Command-J (Mac) or Control-J (Windows/Linux)

What happens when you reject or accept a suggestion

The actions that you take when a recommendation is provided impact the training process of the model. If you hit the **Esc** key to reject a recommendation, then the telemetry process considers that a rejection of the recommendation. Red Hat and IBM will view that action as the user determining that the recommendation was not suitable for their task intent. If you hit **Enter** and accept the recommendation, then the telemetry process considers that action an acceptance of the recommendation. Red Hat and IBM will view that action as the user determining that the recommendation was good enough to use over typing an alternative directly.

If a recommendation is accepted, and then further edits are performed, then the act of changing the recommendation to something else will be considered a modification of the recommendation. This will tell Red Hat and IBM that the recommendation required extra action in order to meet the intended use. This information will be used for context in training the model for similar prompts in the future.

The telemetry data is first anonymized and then sent whenever you switch to a different file in Visual Studio code or create a new Ansible task in the same Ansible Playbook. For more information see [Getting a Recommendation](#).

Improving the recommended guidance

Follow these guidelines to improve the likelihood of a quality recommendation:

- ▶ Ensure that your YAML is properly formatted.
- ▶ Avoid context switching within a single playbook file. Lightspeed attempts to correlate earlier tasks to the active recommendation, and context switching may lead to incorrect recommendations.
- ▶ If you do not get a recommendation that aligns with the intent of your task name, then rewording your statement to provide more information on what is desired may lead to different results. Try adding or removing context to see if you get a better response.

Some example tasks to try out in IBM watsonx Code Assistant for Red Hat Ansible Lightspeed are shown in Example 7-1.

Example 7-1 Example tasks for Ansible Lightspeed

```
- name: Update all packages
- name: Create a user named 'oracle'
- name: Create a 40Gb Logical Volume
- name: Run 'uptime' command on remote servers
- name: Create AIX volume group named datavg
- name: Restart chronyd daemon
- name: Add host entry to /etc/hosts file
- name: Add the line "Defaults logfile=/var/log/sudo.log" to /etc/sudoers
```

For more information see [Improving the Recommended Guidance](#).

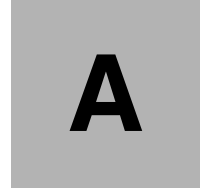
Matching recommendations to training data

The IBM watsonx Code Assistant for Red Hat Ansible Lightspeed machine learning model is trained on content from Ansible Galaxy and other sources.

Given the nature of deep learning technology, as well as the kinds of content used to train and which are generated by Lightspeed, it is not possible to identify specific training data inputs that contributed to particular Lightspeed output recommendations. Nevertheless, Lightspeed includes a feature to help users that are interested in understanding possible origins of generated content recommendations. When Lightspeed generates a recommendation, it will attempt to find items in the training dataset that closely resemble the recommendation. In such cases, Lightspeed will display licensing information and a source repository link for the training data matches in a panel interface in the VS Code extension.

This feature may enable users to ascertain open source license terms that are associated with related training data. This feature has been implemented even though it is believed to be unlikely that either the training data used in fine-tuning or the output recommendations themselves are generally protected by copyright, or output reproduces training data content controlled by copyright licensing terms.

Red Hat does not claim any copyright or other intellectual property rights in the suggestions generated by the IBM watsonx Code Assistant for Red Hat Ansible Lightspeed service. For more information see [Matching Recommendation to Training Data](#).



Unveiling IBM i Merlin

In this appendix, we explore IBM i Merlin, providing insightful descriptions and in-depth discussions on its various facets. The sections presented offer an introduction and comprehensive overview of Merlin's key aspects, benefits, collaboration between IBM and ARCAD, components, content, and its significance in meeting the demands of DevOps on IBM i. This exploration aims to provide a clear understanding of IBM i Merlin and its role in modernizing the IBM i ecosystem.

The following topics are covered in this appendix.

- ▶ Introduction
- ▶ What is Merlin
- ▶ Merlin's problem-solving impact
- ▶ Benefits of Merlin for IBM i modernization
- ▶ Decades of collaboration: IBM and ARCAD
- ▶ Components of Merlin
- ▶ Comprehensive overview of Merlin content
- ▶ Ansible integration for IBM i lifecycle management via Merlin
- ▶ The business demands for DevOps on IBM i
- ▶ Merlin for IBM i developers
- ▶ Merlin requirements

A.1 Introduction

In today's rapidly evolving business landscape, IBM i customers face the pressing need to modernize their applications and stay ahead of the game. As you plan your IT environment, we understand the top concerns that you grapple with, from “cobbling together” various tools to “force-fitting” IBM i native file systems. Many are still reliant on outdated technologies which lack automated change control and project builds, and grapple with monolithic, non-modular designs and ancient source editors.

At IBM, we are committed to guiding the market towards “IBM i Next Gen Apps” – applications that can quickly respond to business needs through DevOps, CI/CD, and Agile methodologies. With a focus on encapsulating processes and data, we help you create assets for the business by blending technology to achieve the best fit for purpose. Moreover, we enable you to easily incorporate new technologies, even if they are not currently “in-house.”

To get to “IBM i Next Gen”, you need to address various challenges, such as:

- Converting fixed-format RPG to free format
- Understanding and managing high volumes of code
- Refactoring mega-programs into modules
- Ensuring intelligent builds amidst spaghetti code.

Exposing embedded logic as services and adopting a “service consumption” mindset and tools are crucial steps forward. Utilizing modern tools such as *Git* for common source code management can be transformative.

Fortunately, IBM i offers a range of modernization technologies to bridge the gap. Modern RPG and its integration with contemporary development tools helps address the talent gap. Connectivity with cloud-based and containerized applications via Rest APIs facilitates smooth communication between systems. We provide ISV and Open Source tools to modernize old source code and fully adopt DevOps practices.

One such tool is IBM i Modernization Engine for Lifecycle Integration (Merlin) – an innovative set of OpenShift-based tools designed to guide and assist software developers in modernizing IBM i applications and development processes. Running in OpenShift containers, IBM i Merlin permits you to unlock the value of hybrid cloud and provides a multi-platform DevOps implementation. The framework simplifies the adoption of DevOps and CI/CD practices, while utilizing technologies that promote services-based software through RESTful interface connections and enterprise message technologies.

The Merlin platform includes IBM i VM management, which provisions, manages, and deletes IBM i virtual machines through PowerVC or PowerVS in IBM Cloud. One of the actions available to run on the IBM i server is *Enable Ansible environment*, which helps initiate the yum, python, and Ansible packages. With IBM i Merlin, we pave the way to the future of application development, propelling you towards the realms of efficiency, agility, and innovation.

A.1.1 What is Merlin

Merlin emerges as a transformative solution for the IBM i platform, driving the critical need for application modernization. This robust framework enables a fluid generational transition, preserving customer investment while propelling the platform's evolution. The framework guides and simplifies the use of the tools which help implement DevOps and continuous integration (CI) and continuous delivery (CD) – also referred to as CI/CD. Merlin embraces

standardization, making it accessible to the younger generation already familiar with these tools.

Central to Merlin's impact is the inclusion of the RPG converter, a pivotal tool enabling the modernization of core RPG code. With this advancement, RPG becomes more appealing and user-friendly, enticing new developers into the fold.

Moreover, Merlin champions cloud infrastructure migration, providing agile Dev and test environments. This paradigm shift offers productivity gains, opening doors to new resources accessible from any location with top-notch security measures.

Note: Upcoming iterations will incorporate supplementary applications driven by feedback from customers and partners, enhancing the platform's responsiveness to user needs.

Figure A-1 illustrates a depiction of the IBM i Modernization Engine for Lifecycle Integration GUI - overview. This interface has been meticulously crafted to enhance the capabilities of IBM i users, allowing interaction with hybrid cloud work tools. These tools facilitate modern development and deployment of IBM i native applications, utilizing standardized cloud methods. The GUI showcases Merlin's commitment to providing accessible and efficient solutions for application modernization and integration in the dynamic IT landscape.

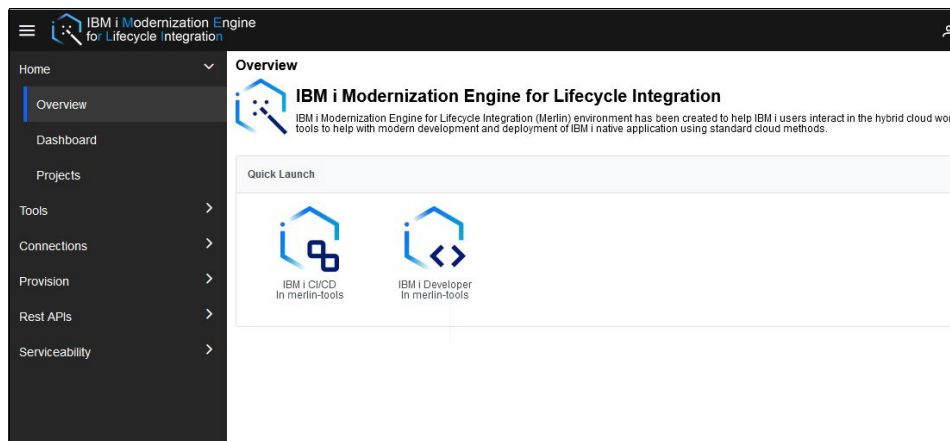


Figure A-1 IBM i Merlin GUI overview

A.1.2 The role of Merlin in the IBM i market

IBM introduces IBM i Merlin - a transformative “Wizard of wizards” operating within containers - to enable IBM i clients with a focused modernization approach. This innovative solution guides clients through the process of smoothly transitioning from outdated, inefficient technology to modern versions of existing technology.

IBM i Merlin strategically emphasizes adopting modern tools and processes such as DevOps, cloud services, and hybrid cloud solutions – propelling the IBM i ecosystem into a new era of efficiency and adaptability. Through the integration of container-based tools, clients gain the agility to keep pace with the ever-evolving demands of the market.

IBM i Merlin takes center stage in the Red Hat OpenShift conversation, positioning IBM i directly at the forefront of modernization discussions. Its containerized architecture opens doors for clients to use the potential of hybrid cloud and multi-platform DevOps implementation to their advantage.

A crucial factor to the development of IBM i Merlin was the active involvement of IBM i customer advisory councils, ensuring that the solution aligns with the specific needs and aspirations of clients. Furthermore, expert minds in the IBM i modernization domain stand ready to assist clients in adopting IBM i Merlin, ensuring an effortless and successful transition.

With IBM i Merlin as your ally, embark on a journey of transformation, bridging the gap between existing systems and cutting-edge technologies. Confidently step into the future of the IBM i ecosystem, and let IBM i Merlin be your guide to a brighter, more agile tomorrow.

A.1.3 Merlin's problem-solving impact

In this section, we explore the remarkable problem-solving capabilities of Merlin, the revolutionary solution for the IBM i platform. Table A-1 shows the profound impact Merlin has across four critical areas:

Table A-1 Key aspects of Merlin's problem-solving impact

Modern / Centralized source control and branching	Modern RPG – modular and free format	Browser-Centric IDE based on VS Code	Application blueprint
Use GitHub, GitLab, Bitbucket or Gitbucket to enhance source control efficiency and facilitate efficient branching processes for improved development workflows.	Transformation of RPG code from fixed to free format for enhanced modularity and readability through refactoring.	Explore Merlin's browser-centric IDE with features such as outline view, tokenization, content assist, code formatting, and language understanding.	Utilize Merlin's capabilities for impact analysis, program understanding, data usage analysis, and program flow visualization, ensuring informed decisions and application integrity.

A.1.4 Benefits of Merlin for IBM i modernization

Within the sphere of IBM i modernization, Merlin emerges as an influential catalyst, orchestrating a range of advantages that amplify development, simplify deployment, and elevate overall operational efficiency. Table A-2 explores the multitude of benefits facilitated by Merlin.

Table A-2 Benefits of Merlin for IBM i modernization

Benefit	Description
Faster provisioning	Experience rapid provisioning and deprovisioning of IBM i development environments, ensuring nimbleness in your development cadence.
Modernized IBM i applications	Automate the conversion of fixed-format RPG code into the more supple free-form RPG, modernizing your applications for heightened legibility and manageability.
Reduced time to market	Expedite the creation of new IBM i business applications and realize swifter deployment, empowering you to promptly address market dynamics and out pace competitors.

Benefit	Description
Single DevOps pipeline	Simplify your application development process through a unified DevOps pipeline that efficiently guides your code from testing to production realms.
Accelerated developer onboarding	Minimize the learning curve for new developers by harnessing modern tools such as Git and Jenkins, ensuring swifter adaptation and heightened productivity.
Cloud enabled	Adopt a hybrid cloud approach and empower your IBM i applications to utilize the potential of multi-platform CI/CD implementation, leading to enhanced scalability and innovation.

A.1.5 Decades of collaboration: IBM and ARCAD

ARCAD, a trailblazer in DevOps for IBM i, has forged an enduring partnership with IBM spanning over two decades. Throughout their journey together, several notable milestones mark their collaborative efforts.

In 2003, ARCAD achieved integration with WDSC (predecessor of RDi), laying the groundwork for their future endeavors. Four years later, in 2007, they introduced the RPG Free Form converter, revolutionizing RPG development on IBM i. Their commitment to innovation was recognized in 2012 when ARCAD was awarded the prestigious IBM Rational® Innovation Award.

The collaboration deepened further in 2013 with ARCAD licenses becoming available in IBM Passport Advantage®, completing RTC integration. In 2016, ARCAD smoothly integrated with Urbancode, enhancing their DevOps capabilities for IBM i. The following year, in 2017, ARCAD Observer and RPG converter found their way into the e-config Channel, expanding accessibility to these powerful tools.

Fueling their success is ARCAD's steadfast dedication to continuous R&D investment in DevOps for IBM i, spanning eight years. They proudly boast a globally diverse team, contributing a rich cultural perspective to their work. ARCAD's integration with industry-leading DevOps flagship products such as Git (GitHub, Bitbucket, Gitlab, RTC), Jenkins, and Jira reinforces their position as the preferred choice for the largest IBM i organizations, solidifying their reputation as a trusted industry reference in IBM i DevOps.

Note: IBM and ARCAD Software have had a long-standing partnership. ARCAD had previously created plugins to the architecture being designed. To deliver the best value to the market as fast as possible, IBM chose to work with ARCAD to deliver a product with an integrated RPG modernization and impact analysis.

A.1.6 Components of Merlin

In the domain of IBM i modernization, Merlin acts as a potent catalyst. It introduces a collection of robust components that reshape the realm of application development. These components are purposefully crafted to facilitate smoother workflows, foster better collaboration, and embrace contemporary software practices. In this section, we explore the pivotal components of Merlin, describing their significance in propelling IBM i into a new era of innovation.

The following individual components empower Merlin to excel in both problem-solving and modernization:

▶ Eclipse Theia

At the heart of Merlin's development environment resides Eclipse Theia, an open-source iteration of Microsoft's original Visual Studio Code (VS Code). This dynamic platform offers a versatile and user-friendly integrated development environment (IDE) for crafting and refining IBM i applications. For further about IDE take a look at “Summary of the Integrated Development Environment (IDE)” on page 337

▶ Eclipse Che

A pivotal element in Merlin's infrastructure, Eclipse Che provides the workplace server responsible for crafting, managing, and orchestrating the IDE within a Kubernetes environment. This integration ensures a fluid and efficient development process which is further enhanced by Kubernetes orchestration.

▶ ARCAD Transformer

Integral to the Merlin ecosystem, Transformer – formerly known as ARCAD Converter – exemplifies the powerful software that facilitates the conversion and transformation of existing code into more contemporary and efficient forms. This essential tool simplifies the modernization process and contributes to the advancement of IBM i applications.

▶ ARCAD Builder

A cornerstone of Merlin's capabilities, ARCAD's build management software (Builder) empowers developers with advanced tools for efficiently assembling and managing application components. This software promotes consistency, reliability, and efficient deployment practices throughout the development lifecycle.

▶ ARCAD Observer

Within Merlin's toolkit, the Observer part of ARCAD emerges as a noteworthy software component. This tool provides comprehensive insights into the application development and deployment processes, offering valuable visibility and control over critical aspects of the development lifecycle.

▶ Integration with Git

Merlin integrates with Git, a widely adopted version control system, enhancing collaboration and code management. This integration streamlines code repository operations and aligns Merlin's capabilities with modern development practices.

▶ Jenkins

A crucial component, Jenkins facilitates continuous integration and continuous deployment (CI/CD) pipelines, a central focus of Merlin's development approach. IBM has incorporated Jenkins as part of Merlin, enhancing the platform's ability to automate and optimize the application delivery pipeline.

▶ Red Hat OpenShift

Merlin finds its home and operational foundation within Red Hat OpenShift, a robust container platform. OpenShift empowers Merlin to be installed, managed, and executed efficiently, supporting its role as a transformative solution for IBM i modernization.

▶ IBM Cloud Pak foundational services

IBM Cloud Pak® foundational services constitute the cornerstone of the Cloud Pak ecosystem, encompassing critical tools such as Certificate Manager for efficient certificate administration, enabling secure connections, and License Manager for centralized software entitlement tracking, enhancing licensing efficiency. These services underscore a commitment to a robust and secure cloud environment.

Note: Customers do not need to pay extra to acquire the ARCAD functions. These functions are fully integrated into the Merlin product. As a result, developers have complete access to “Fixed to Free” format conversion, an integrated impact analysis tool, and the capability to utilize intelligent build support directly within the IDE. These valuable features, powered by ARCAD, are fully integrated and included as essential components of the Merlin solution.

Summary of the Integrated Development Environment (IDE)

In the context of IBM i modernization, an integral aspect lies in the robust Integrated Development Environment (IDE) that effectively combines offerings from both IBM and ARCAD. This cohesive environment not only enhances the development process but also fosters efficient modernization of applications.

Table A-3 presents a comprehensive overview of the Integrated Development Environment (IDE), highlighting its diverse features and capabilities that empower developers on the journey towards innovation.

Table A-3 Comprehensive overview of the Integrated Development Environment (IDE)

IBM i Integration	ARCAD Integration
Integrated without disruption with Code Ready Work Spaces.	Efficient Git repository setup, enabling code migration from previous library to Git.
VS Code plugins offering features such as tokenization, color coding, outline view, content assist, refactoring, intelligent formatting, and more.	Intelligent build metadata population.
Robust project explorer facilitating efficient IBM i environment and source management.	Conversion of code to Fully Free form, utilizing deep expertise in transitioning source control from existing to Git-centric.
Intelligent build with integrated compile feedback, defined metadata, and a comprehensive joblog explorer.	ARCAD dependency-based build, addressing the complexities of IBM i applications through automated tooling and processes.
Git integration for utilizing Git-based tooling, encompassing actions such as pull, push, and merge.	ARCAD impact analysis, offering valuable insights into application linkages, data usage, and code flow visualization.

A.1.7 Comprehensive overview of Merlin content

Merlin's enhanced interface is highlighted on the left-hand side, showcasing the components of the modernization engine within the yellow box (refer to Figure A-2 on page 338). This interface features the IDE and CI/CD components, providing a user-friendly and cohesive environment. Recognizing the learning curve for IBM i customers new to Linux tools and CI/CD, the platform was designed with simplicity and integration in mind. It aims to preserve the IBM i approach, enabling a smooth transition to Linux and CI/CD for IBM i users. The framework works wherever Linux does, be it on-premises, within IBM Power Systems, or other configurations. Regardless of the setup, a user-friendly browser interface ensures a unified experience. This required creating a customized graphical user interface and an engine facilitating effective communication between code components.

Key attributes:

- ▶ A modernization platform guiding IBM i applications toward hybrid DevOps.
- ▶ Exposing IBM i native functions via Restful interfaces and centralizing IBM i connections management.
- ▶ Facilitating the use of tools for DevOps and services-based software implementation.

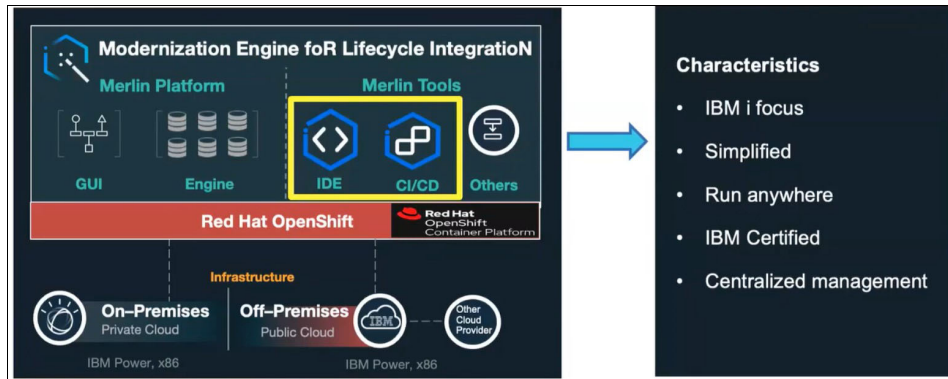


Figure A-2 Graphical view of Merlin

Deploying the Merlin Platform through the OpenShift Web Console

In terms of visibility, this process is mostly transparent to users. However, the underlying mechanism involves fitting Merlin into the deployment and acquisition strategies, aligning with the procurement and deployment of container-based applications from IBM. The OperatorHub serves as the platform where you can access and download Merlin, while also facilitating the necessary purchases.

This process is visually depicted in Figure A-3, where the OperatorHub under Red Hat OpenShift for Merlin is displayed.

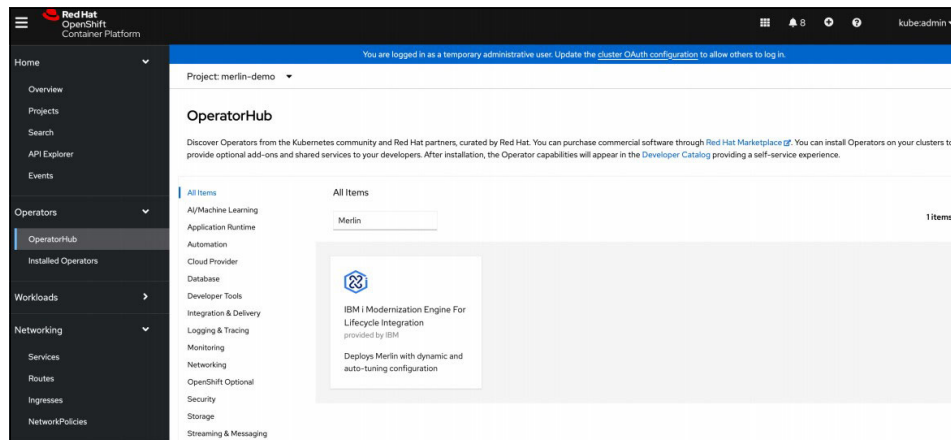


Figure A-3 OperatorHub integration with Red Hat OpenShift for Merlin

Overview of the IBM i Platform

Once deployed, a comprehensive suite of components coalesce to form a cohesive project and product. This suite includes user management, activity monitoring, and authorization enforcement, delivering the expected integrated experience of an IBM i product. What sets this apart is that it is orchestrated by a set of containers, orchestrating the IDE, CI/CD, and forthcoming elements.

In Figure A-4 you can observe the extensive capabilities of the Merlin platform. These includes the Merlin tool lifecycle, authentication, certification management, user management, monitoring, inventory management, credential management, IBM i VM management, and IBM i software installer.

Moreover, the Merlin layer itself showcases the Merlin platform's composition, which includes both the GUI and engine. The Merlin tools are further depicted, incorporating the IDE, CI/CD, and other essential elements.

Furthermore, the infrastructure is depicted, demonstrating where Red Hat OpenShift operates, whether on-premises or in the cloud, in ppc64 or x86 environments.

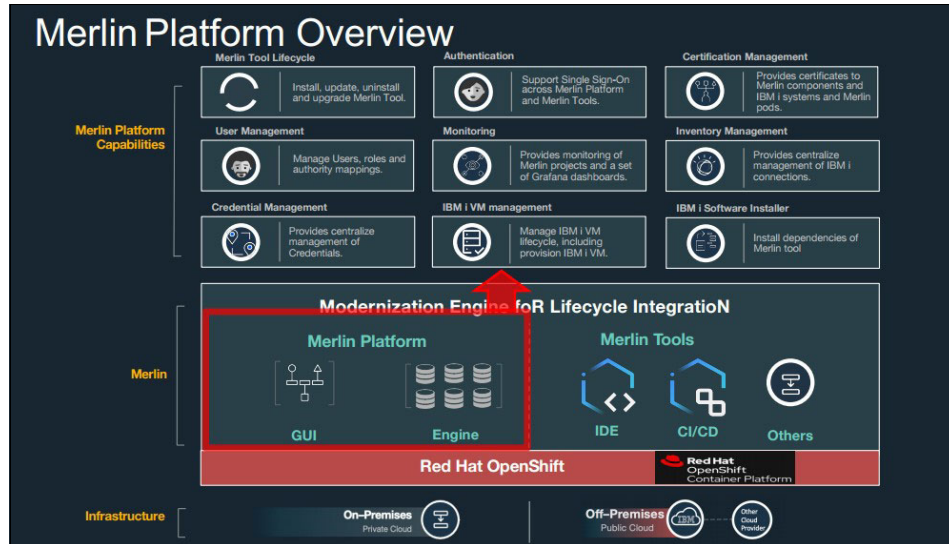


Figure A-4 Merlin platform overview

Merlin platform: Creating RESTful services for IBM i systems

Merlin platform offers a phased approach to generate RESTful services from your existing assets. This allows you to target specific components for transformation. Once these components are prepared, they can be converted into services. Merlin understands the intricacies of modernization, aiming to simplify your journey.

Key points:

1. User-friendly GUI for REST services: Merlin's platform provides an intuitive interface for launching RESTful service creation.
2. Develop RESTful services: Create RESTful services for IBM i programs and data on IBM i systems in the initial release.
3. Native IBM i execution: RESTful services continue to run natively on IBM i.
4. Wide language support: Support for RPG, COBOL, and program/service program (PGM/SRVPGM).
5. Data Integration: Integrate data stored in DB2 for i into your RESTful services.

Figure A-5 shows the creation of a Web Services server based on IBM i objects including RPG and COBOL programs, alongside SQL statements.

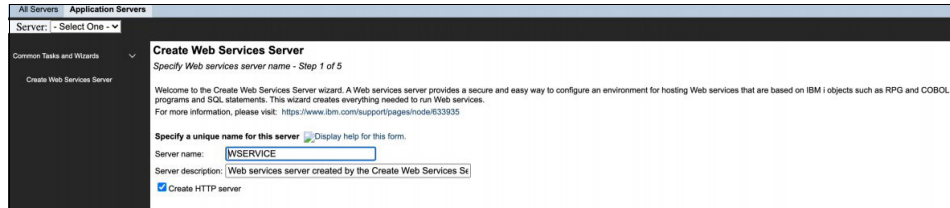


Figure A-5 Web Services server creation on IBM i objects

Ansible integration for IBM i lifecycle management via Merlin

The lifecycle management of the product itself is adeptly handled, extending to deployment options. For instance, deployment can occur in IBM Power Systems Virtual Server, which operates on IBM Cloud. Alternatively, you can deploy within your own infrastructure, using IBM PowerVC on your server. This flexibility ensures accessibility, whether you are transitioning to hybrid cloud, fully embracing the cloud, or maintaining an on-premises setup. Irrespective of the choice, Merlin's CI/CD and IDE capabilities remain versatile and accessible.

IBM i virtual machine provisioning with Ansible automation:

- ▶ Supports PowerVC and PowerVS environments.
- ▶ Direct VM provisioning from templates, facilitating integration with CI/CD tools.
- ▶ Enables software installation on IBM i platforms, aligning with hybrid cloud strategies.

Merlin incorporates specialized Ansible playbooks for PowerVC and PowerVS environments. While not the mainstream approach due to often static IBM i LPAR structures, administrators can enable this functionality. This process involves:

1. Initiate the process by crafting a PowerVC template within the Inventory. Figure A-6 shows a visual depiction of the **Inventory** interface, which facilitates this essential step in the workflow.

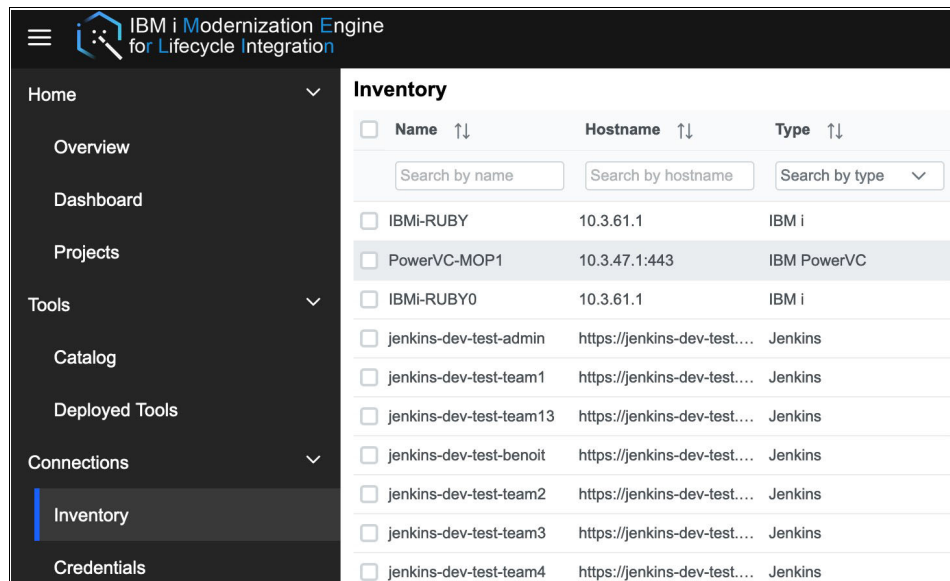


Figure A-6 Creating a PowerVC template in the Inventory

- In this step, you configure the PowerVC credentials, enabling Merlin to securely communicate with the PowerVC instance. This involves providing the necessary authentication details to establish a connection. Figure A-7 shows editing the **Inventory**.

Edit Inventory

*Name:

Type:

*Hostname:

*Port:

Owner:

Description:

Figure A-7 Editing Inventory to Configure PowerVC Credentials

- In this step, leverage Merlin's intuitive GUI to initiate VM provisioning which is working with a Merlin **Template**. This process simplifies the creation of virtual machines, facilitating the deployment of IBM i instances on PowerVC or PowerVS. Follow the on-screen instructions as shown in Figure A-8 and input the necessary details to customize your VM's configuration.

Add Template

Basic Configuration

*Name:

*Inventory:

*Credential:

Type:

Owner:

Description:

Virtual machine Configuration

*Virtual machine name:

*Compute template:

*Source image:

*Primary network:

Run CL command:

Figure A-8 Adding a Template in Merlin's GUI for VM provisioning

- After completing the configuration, a dedicated Merlin menu becomes available within the GUI, offering access to the VM provisioning process. This feature transforms Merlin into a central hub, that permits developers to provision PowerVC or PowerVS VMs tailored for modernization projects. New VMs integrate dynamically into the Inventory for use by 'IBM i Developer' or 'CI CD' services. In Figure A-9 on page 342 go to **Provision -> Deploy Virtual Machine**.

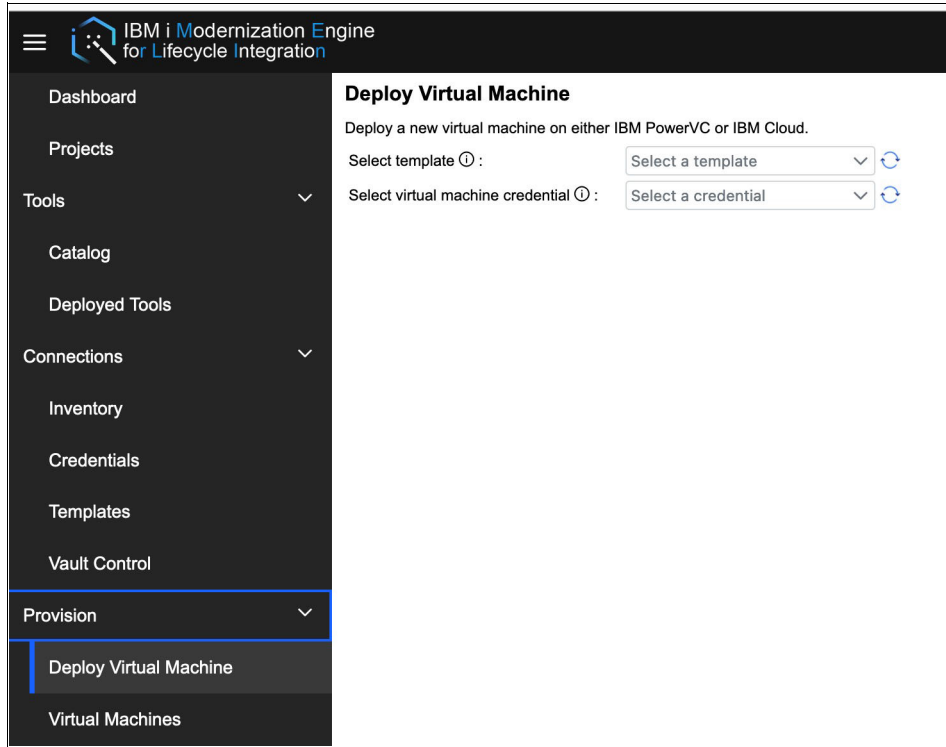


Figure A-9 Navigating the Deploy Virtual Machine menu in Merlin's GUI

Note: Similar steps above apply to PowerVS provisioning, requiring an IBM Cloud API Key for credentials. Refer to [Manage IBM i Servers](#)

Within the Merlin pod, a set of Ansible playbooks and the Ansible engine facilitate internal setup, initialization of the Merlin-IBM i environment, and PowerVS or PowerVC provisioning.

Note: It is worth noting that Merlin does not incorporate Terraform. This signifies that Terraform is not utilized as an automation tool for provisioning IBM i VMs within Merlin. Instead, the automation tool employed is Ansible.

Merlin version 1.0 introduces an Ansible controller integrated into the Merlin engine. This controller orchestrates:

- ▶ Internal playbooks designed for IBM i VM provisioning using PowerVC or PowerVS through the use of OpenStack and IBM Cloud modules. This includes two key playbooks: VM Provisioning and VM Destroy.
- ▶ A default set of six playbooks which are referred to as 'actions' in Merlin.

Administrators are required to execute these actions in a sequential manner to prepare the target IBM i LPAR for efficient management by Merlin. These actions pave the way for development with Merlin, build processes, and CI/CD practices.

The six actions provided by Merlin are:

- i. Enabling Ansible: Installs essential packages such as yum, python, and Ansible on the IBM i server.
- ii. Validating PTF Level: Verifies the PTF (Program Temporary Fix) level using Merlin.

- iii. Installing Certificates: Facilitates secure communication by installing necessary certificates.
- iv. Enabling IBM i developer: Enables the IBM i developer environment.
- v. Enabling remote debugger: Permits administrators to enable the remote debugger feature.
- vi. Enabling ARCAD environment: Installs ARCAD solutions on the target IBM i system.

These are shown in Figure A-10.

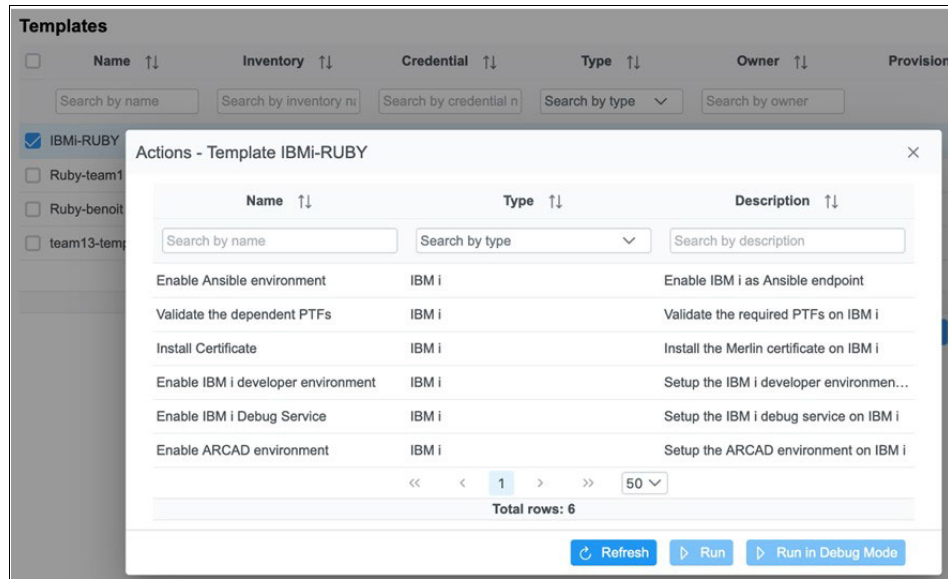


Figure A-10 Six actions on IBM i performed from Merlin by the administrator.

Note: In future releases, based on the evolving product roadmap which is subject to potential changes, additional playbooks will be introduced. These forthcoming playbooks will utilize tasks such as PTF management, Security & Compliance management, and more, utilizing Ansible to effectively manage IBM i environments. However, such additional playbooks are not included in version 1.0, reflecting Merlin's aim to offer user-friendly and supported automation even for those with limited Ansible knowledge or skills.

A.1.8 The business demands for DevOps on IBM i

The path to modernization in the ever-evolving technological landscape calls for a comprehensive approach. Here are essential steps to revamp your development processes and embrace the principles of DevOps:

- First, encourage a natural skills transfer by fostering a mindset shift and cultivating champions within your teams, ensuring effective knowledge dissemination. Next, prioritize security by implementing new tools that meet critical requirements and elevate your overall security measures.
- Finally, address complex application architectures by utilizing specialized tools to re-factor them, transitioning towards more flexible and services-based structures, which enables easier updates and modernization.

Note: By following these guidelines, your organization can make significant strides towards a more efficient and innovative future.

DevOps MVP architecture overview - preceding Merlin

In the world of DevOps environments, the landscape preceding the advent of Merlin often featured intricate setups. Many clients have initiated the process of setting up DevOps frameworks using tools such as Jenkins, Ansible, and others. Such initiatives aligned well with the prevalent practices in their software development and deployment across different platforms.

However, for the IBM i ecosystem, these attempts resulted in a force-fit scenario, as the unique requirements of the IBM i platform needed to be integrated within these existing frameworks. Figure A-11 illustrates the complexities and challenges faced by those striving to align their processes. This diagram showcases the DevOps landscape prevalent before Merlin's introduction, emphasizing the need for a more tailored solution.

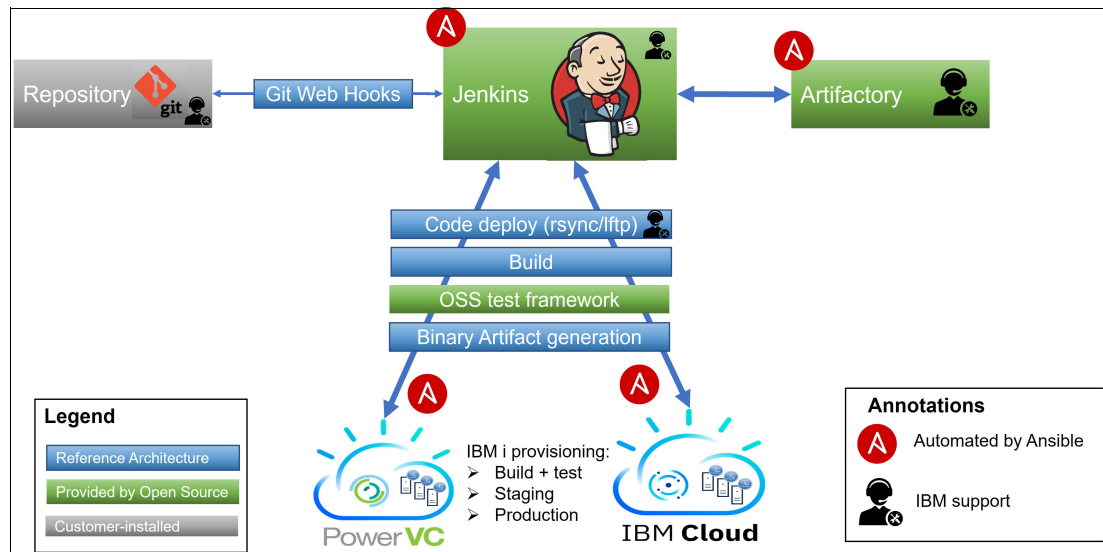


Figure A-11 DevOps MVP Architecture and integration diagram before Merlin

Full cycle of the CI/CD process on IBM i with Ansible

This section explores an in-depth understanding of the DevOps pipeline process, commencing with continuous integration and continuous deployment. We illustrate this process using various tools and solutions, including Git, Ansible, and IBM PowerVC (OpenStack), among others.

Figure A-12 on page 345 depicts the architecture constructed. It is important to note that the primary focus is on continuous integration (CI), although continuous deployment (CD) require additional workflows.

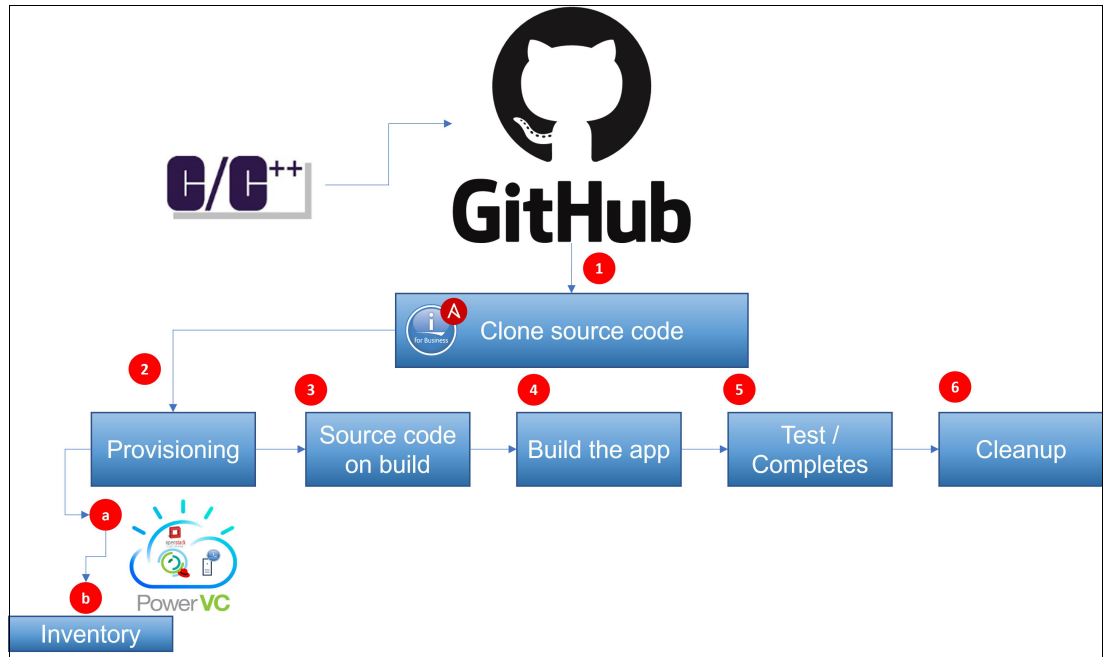


Figure A-12 End-to-end CI/CD process diagram for IBM i with Ansible

Now examine the playbooks central to this use case. The set of playbooks, along with an inventory file and the Ansible configuration file, is outlined in Example A-1.

Example A-1 Set of playbooks to run full cycle of the CI/CD process on IBM i

```

|-- add_build_system.yml
|-- ansible.cfg
|-- build.yml
|-- cleanup.yml
|-- git_clone.yml
|-- hosts.ini
|-- main.yml
|-- post_build_actions.yml
|-- provision_vars.yml
|-- provision_vm.yml
|-- put_code.yml

```

The core playbook for CI/CD is labeled as `main.yml`. Example A-2 offers a clear insight into the upcoming execution process.

Example A-2 Principal playbook for CI/CD named `main.yml`

```

---
- hosts: localhost
  vars:
    build_with_stmfs: true
    provision: true
    cleanup: true
  vars_prompt:
    - name: build_number
      prompt: "Enter a build number"
      private: no
    - name: git_repo_url
      prompt: "Enter a git repo URL"
      private: no

```

```

    - name: git_branch
      prompt: "Enter a git branch"
      private: no
collections:
  - ibm.power_ibmi
tasks:
  - set_fact:
      build_lib: "BUILD_{{ build_number }}"

  - set_fact:
      build_path: "/tmp/{{ build_lib }}"
      local_workspace: '~/workspace/{{ build_lib }}'

  - block:
      - name: Step 1 - clone source code from git
        include: git_clone.yml

      - block:
          - name: include provision related vars if provision is true
            include_vars: provision_vars.yml

            - name: Step 2.1 - provision vm on demand
              include: provision_vm.yml
              when: provision

          - name: Step 2.2 - add build system to in-memory inventory
            include: add_build_system.yml

          - name: Step 3 - put source code to build machine
            include: put_code.yml

          - name: Step 4 - build your app on build machine
            include: build.yml

          - name: Step 5 - run test and completes
            include: post_build_actions.yml
            delegate_to: "build_system"

  always:
    - name: Step 6 - cleanup on demand
      include: cleanup.yml
      when: cleanup
...

```

The steps outlined in the `main.yml` (as shown in Example A-2 on page 345) call distinct YAML files. These steps are:

1. **Clone source code:** When exploring the realm of CI/CD, the underlying objective remains consistent. Essentially, you initiate a pipeline with an input – often your source code – and as this section progresses, an outcome is generated, which can be a packaged program. Consider tasks akin to a “git clone.” Consequently, it is likely that the initial step in your pipeline predominantly involves cloning.

The subsequent YAML file, dedicated to this task, follows a sequence:

- If a local workspace already exists on the system, it is removed.
- Following this, a new local workspace is created at the localhost.
- Subsequently, the action involves cloning a git repository at the localhost, where the localhost functions as the IBM i control node in this context.

The requisites for this action include the repository URL, the previously created local workspace, and the designated `git` branch.

Notably, the variables pertaining to this process are specified within the `main.yml` file. The YAML file dedicated to the cloning process is shown in Example A-3.

Example A-3 Playbook for clone source code

```
---
- name: remove if {{ local_workspace }} exists
  file:
    path: '{{ local_workspace }}'
    state: 'absent'

- name: create {{ local_workspace }}
  file:
    path: '{{ local_workspace }}'
    state: 'directory'
    mode: '0755'

- name: git clone from source repository
  git:
    repo: '{{ git_repo_url }}'
    dest: '{{ local_workspace }}'
    version: '{{ git_branch }}'
...
```

2. **Provisioning:** The provisioning phase emerges as a crucial cornerstone of the overall process. This segment encapsulates two distinct elements, each playing a significant role in the smooth configuration and deployment of the IBM i virtual machine. These components can be delineated as follows:

- a. **Provisioning variables:** This YAML file serves as a repository for predefined variables utilized in provisioning the IBM i virtual machine. It is essential to bear in mind that this process entails PowerVC-related information. However, a relevant aspect is that the majority of variables within this context remain static, eliminating the need for further modifications.

Additionally, take note of the creation of a fresh user profile designated as “BUILDER” via cloud-init. The structure of this file is shown in Example A-4.

Example A-4 Playbook repository for predefined variables for provisioning

```
---
powervc_host: "192.168.1.101"
powervc_admin: "root"
powervc_admin_password: "abc123"
project: ibm-default
project_domain: Default
user_domain: Default
vm_name: "VM-{{ build_lib }}"
verify_cert: false
image_name_or_id: "IBMi_73"
nic_list: [{ 'net-name': 'Network1' }]
flavor_name_or_id: tiny
deploy_timeout: 300
deploy_userdata: |
  {%- raw -%}#!/bin/sh
  mkdir /home/BUILDER
  system "CRTUSRPRF USRPRF(BUILDER) PASSWORD(abc123) USRCLS(*SECOFR) ASTLVL(*SYSVAL)
  TEXT('Ansible CICD build') HOMEDIR('/home/BUILDER)'"
```

```

system "chgtcpsvr svrspcval(*sshd) autostart(*yes)"
system "strtcpsvr *sshd"
{% endraw %}
...

```

- b. **Provision virtual machine:** This YAML file introduces the inclusion of a PowerVC host into the in-memory inventory, effectively supplanting the manual approach of editing the inventory file for host addition. In the process of provisioning, the **os_server** compute instance from OpenStack is employed. Post-provisioning, an additional task is executed to sift through the output, thereby revealing the IP address of the freshly provisioned IBM i virtual machine. This is shown in Example A-5.

Example A-5 Playbook that introduces a PowerVC host into the in-memory inventory

```

---
# Add powervc host to in-memory inventory
- name: Add PowerVC host {{ powervc_host }} to Ansible in-memory inventory
  add_host:
    name: 'powervc'
    ansible_user: '{{ powervc_admin }}'
    ansible_ssh_pass: '{{ powervc_admin_password }}'
    ansible_ssh_extra_args: -o StrictHostKeyChecking=no
    ansible_python_interpreter: /usr/bin/python3
    ansible_ssh_host: '{{ powervc_host }}'
  no_log: true

# New vm info from OpenStack
- name: Deploy a new VM
  os_server:
    auth:
      auth_url: https://{{ ansible_ssh_host }}:5000/v3
      username: '{{ ansible_ssh_user }}'
      password: '{{ ansible_ssh_pass }}'
      project_name: '{{ project }}'
      project_domain_name: '{{ project_domain }}'
      user_domain_name: '{{ user_domain }}'
    name: '{{ vm_name }}'
    image: '{{ image_name_or_id }}'
    flavor: '{{ flavor_name_or_id }}'
    verify: '{{ verify_cert }}'
    nics: '{{ nic_list }}'
    timeout: '{{ deploy_timeout }}'
    userdata: '{{ deploy_userdata }}'
  register: vm_info
  delegate_to: 'powervc'

- name: New IBM i VM's IP
  debug:
    msg: "{{ vm_info.server.accessIPv4 }}"
...

```

3. **Add build system:** This YAML file introduces the IBM i VM deployed in-memory inventory. Notably, the inventory is populated with values through the utilization of **set_fact**. The IP address of the IBM i VM, obtained from the register during deployment (**vm_info.server.accessIPv4**), is a key inclusion.

Additionally, values from variables such as **ansible_ssh_user** and **ansible_ssh_pass**, which are defined in the **hosts.ini** file, are integrated. Subsequently, the **known_hosts** module plays a role in adding or removing the host key (SSH) for the deployed IBM i VM.

This key is essential for the control node to manage the new virtual machine via Ansible. It is relevant that the term “non-fixed” refers to a new VM, which requires verification of the program (PGM).

Another crucial module comes into play: **wait_for_connection**. This module monitors the new VM's status until it successfully establishes an SSH connection. It is important to recall that the managed node necessitates Python 3 and associated packages.

Consequently, these prerequisites are installed using the **raw** module. The structure of the YAML file is shown in Example A-6.

Example A-6 Playbook that introduces the IBM i VM deployed in-memory inventory.

```

---
- block:
  - name: set_fact for non-fixed build environment
    set_fact:
      build_system_ip: "{{ vm_info.server.accessIPv4 }}"
      build_system_user: '{{ hostvars["non-fixed"]["ansible_ssh_user"] }}'
      build_system_pass: '{{ hostvars["non-fixed"]["ansible_ssh_pass"] }}'

  - name: remove existing entry for vm in case ssh header change occurs.
    known_hosts:
      name: "{{ build_system_ip }}"
      path: ~/.ssh/known_hosts
      state: absent

  - name: add vm-{{ build_lib }} to ansible in-memory inventory
    add_host:
      name: build_system
      ansible_ssh_host: '{{ build_system_ip }}'
      ansible_user: '{{ build_system_user }}'
      ansible_ssh_pass: '{{ build_system_pass }}'
      groups: build_systems
      ansible_ssh_extra_args: -o StrictHostKeyChecking=no
      ansible_python_interpreter: /Q0pensys/pkgs/bin/python3

  - name: wait until vm-{{ build_lib }} is up and ssh ready
    wait_for_connection:
      sleep: 10
      timeout: 1800
      delegate_to: "build_system"

  - name: install python3 on VM-{{ build_lib }}
    raw: /Q0pensys/pkgs/bin/yum install -y python3 python3-itoolkkit python3-ibm_db
    delegate_to: "build_system"
when: provision
...

```

4. **Put code:** This YAML file orchestrates a sequence of tasks essential for code deployment. Initially, the **ibmi_c1_command** module is employed to establish a library at the new virtual machine. This operation is delegated to the IBM i controller node. The subsequent task centers around the **.netrc** file, housing login credentials for authorized access to the newly created IBM i VM. This file resides in the home directory of the IBM i controller node.

A set of tasks ensue within a **block** structure, focusing on path creation. The **ansible.builtin.file** module is employed in this process to enable the creation of directories, guided by the specified variables from **main.yml**. Notably, this approach encompasses subdirectories as required, ensuring a comprehensive directory structure.

The ensuing task is centered on the transfer of the 'C' program from the IBM i control node to the new IBM i virtual machine. To facilitate this transfer and eliminate interactive prompt passwords, the task utilizes the `sshpass` utility. The structure of the YAML file is provided in Example A-7.

Example A-7 Playbook for orchestration the tasks for code deployment

```
---
- name: Create {{ build_lib }} on {{ build_system_ip }}
  ibmi_cl_command:
    cmd: CRTLIB {{ build_lib }}
  delegate_to: "build_system"

- name: "check if ~/.netrc contains ibm i target login"
  lineinfile:
    name: ~/.netrc
    line: "machine {{build_system_ip}} login {{build_system_user}} password
  {{build_system_pass}}"
    state: present
    check_mode: false

- block:
  - name: Create {{ build_path }} on remote IBM i
    ansible.builtin.file:
      path: "{{ build_path }}"
      state: "directory"
    delegate_to: "build_system"

  - name: combine transfer_command
    set_fact:
      transfer_command: "scp {{ local_workspace }}/sendMsg.c
  {{build_system_user}}@{{build_system_ip}}:{{ build_path }}/sendMsg.c"

  - name: put STMFs to remote IBM i
    shell:
      cmd: 'sshpass -p "{{ build_system_pass }}" {{ transfer_command }}'
    when: build_with_stmfs
...

```

5. **Build:** This YAML file orchestrates the execution of crucial tasks within the build process. Here, the `ibmi_cl_command` modules come into play, specifically for invoking the Create Bound C++ Program (`CRTBNDCPP`) command, thereby initiating the ILE C++ compiler. This operation incorporates the utilization of specific variables, as defined in `main.yml`, for the purpose of parameterizing the IBM i command. Notably, `build_lib` and `build_path` are among the variables employed.

Of significance is the use of the source stream file (`SRCSTMF`), which accommodates the program's source code. This code is initially cloned from the Git repository and subsequently transferred to the newly created IBM i VM. Specifically, the program source file named `sendMsg.c` is involved in this process. Upon compilation, an ILE C++ program object named `SENDMSG` is generated.

Example A-8 shows the structure of the YAML file providing insight into the build process.

Example A-8 Playbook to orchestrate the execution of tasks within the build process

```

---
- block:
  - name: call CL command to build application
    ibm.power_ibmi.ibm_i_cl_command:
      cmd: CRTBNDCPP PGM({{ build_lib }}/SENDMSG) SRCSTMF('{{ build_path }}/sendMsg.c')
    when: build_with_stmfs
    delegate_to: 'build_system'
  ...

```

6. **Post-build actions:** In this section, the focus shifts to the execution of essential tasks following the build process. The pivotal task encompasses the initiation of the **SENDMSG** program, followed by the registration of the subsequent output task. This outcome is then systematically filtered to present the outcome resulting from the program invocation.

It is noteworthy to highlight the utilization of a conditional **'when'** directive within this context. Specifically, the **'when'** directive is employed to evaluate a predefined condition, denoted as **'true'** within **main.yml**. This conditional assessment serves as the enabling factor for the execution of the task program, ensuring its activation in the appropriate scenario.

The structure and sequence of tasks pertaining to post-build actions are outlined in Example A-9.

Example A-9 Playbook for running built programs with Stream Files (STMFs)

```

---
- name: run PGM built with STMFs
  ibm.power_ibmi.ibm_i_cl_command:
    cmd: CALL {{ build_lib }}/SENDMSG
    joblog: true
  register: callpgm
  when: build_with_stmfs
- name: PGM output
  debug:
    var: callpgm.job_log[0].MESSAGE_TEXT
  ...

```

7. **Cleanup:** In this YAML file (shown in Example A-10), the process involves the removal of the local workspace and directories from the newly created IBM i VM. Additionally, it encompasses the deletion of the IBM i VM, which was utilized to test the program. Notably, the **pause** module, coupled with a **prompt**, is employed to ensure that the cleanup tasks proceed only upon pressing the Enter key.

Example A-10 Playbook for cleanup and virtual machine deletion

```

---
- pause:
  prompt: Please confirm you want to cleanup! Press enter to continue

- name: remove "{{ local_workspace }}"
  file:
    path: '{{ local_workspace }}'
    state: 'absent'
  ignore_errors: true

- name: Destroy VM when provision
  os_server:
    auth:

```

```

auth_url: https://{{ ansible_ssh_host }}:5000/v3
username: '{{ ansible_ssh_user }}'
password: '{{ ansible_ssh_pass }}'
project_name: '{{ project }}'
project_domain_name: '{{ project_domain }}'
user_domain_name: '{{ user_domain }}'
name: '{{ vm_name }}'
state: 'absent'
delegate_to: 'powervc'
when: provision
...

```

Note: Before the advent of Merlin, DevOps environments often involved intricate setups using tools such as Jenkins and Ansible. While effective for various platforms, integrating IBM i requirements posed unique challenges. The DevOps MVP architecture overview, explored through steps such as clone source code, provisioning, and more, highlights the complexities faced in harmonizing processes. This retrospective underscores Merlin's role in offering a more tailored and efficient IBM i DevOps solution.

Merlin DevOps CI/CD view

The foundation of Merlin's DevOps CI/CD view stems from a visionary concept. By integrating ARCAD into the Merlin product, it efficiently conceals the intricacies, offering a unified solution steered by the Merlin user interface and jointly supported by ARCAD and IBM. This collaborative approach harnesses the collective expertise of both entities, resulting in a cohesive and comprehensive offering.

Constructing this design necessitated a thoughtful consideration of the user ecosystem, focusing on the individuals actively engaging with this transformative product. In the forthcoming section, we show you a thorough exploration of Merlin's DevOps CI/CD view, gaining insights into its intricate aspects and illuminating the collaborative energy that drives this innovative fusion.

The focal point of attention revolves around DevOps, a pivotal paradigm driving the continuous integration and delivery (CI/CD) pipeline. Within this domain, a meticulously crafted suite of tools has emerged, centered on Git and Jenkins. These tools are purposefully designed to strengthen the IBM i platform by facilitating a dynamic approach to continuous development. This involves the automated compilation of code segments extracted from RPG or COBOL applications, subsequently deploying them to diverse IBM i endpoints. The orchestration of these tasks is expertly managed by Jenkins. For a visual depiction, consult Figure A-13.

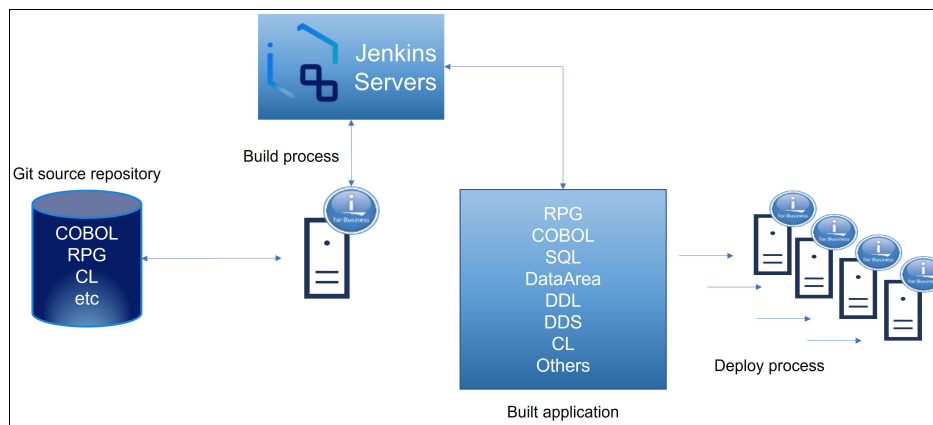


Figure A-13 Enhancing IBM i platform with Git and Jenkins in DevOps CI/CD pipeline

Furthermore, the collaborative partnership between ARCAD and IBM has yielded a deep understanding of the specific needs and intricacies of the IBM i platform. This knowledge has been instrumental in fine-tuning the integration of ARCAD's solutions with Merlin, ensuring a harmonious alignment with IBM i requirements. The robustness of this collaboration is exemplified by ARCAD's suite of plugins that interact with Merlin's capabilities, facilitating a well-integrated and efficient development experience. The resulting synergy between ARCAD's expertise and Merlin's capabilities empowers organizations to achieve optimized DevOps practices and realize the full potential of their IBM i investments. Refer to Figure A-14 for a visual representation of this enriching collaboration.

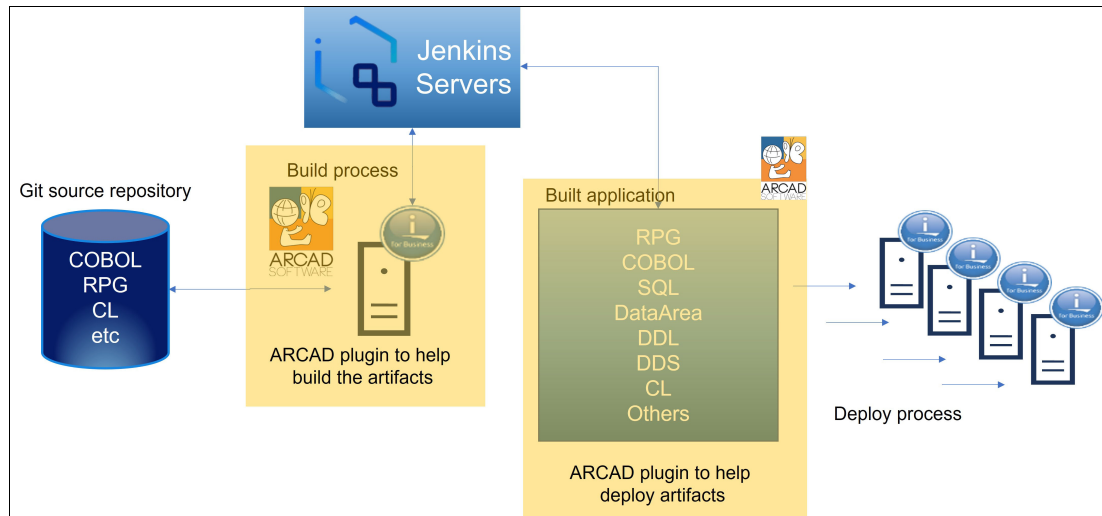


Figure A-14 Enriching the DevOps landscape: ARCAD's integration with Merlin

Merlin architecture on OpenShift Container Platform

In the context of the Merlin architecture on the OpenShift Container Platform, an important addition enters the picture. This new facet entails the role of an OpenShift environment manager. The goal is to facilitate a straightforward learning curve for those who lack an existing OpenShift administrator within their setup. The objective is to impart essential knowledge about setting up an OpenShift environment and integrating Merlin into it without any hindrance. This process simplifies the installation of Merlin within the OpenShift environment, akin to installing software on a developer's PC. However, the distinction lies in the installation occurring on the designated server platform, allowing team members to access it conveniently through their web browsers. As depicted in Figure A-15, an OpenShift administrator assumes the responsibility of overseeing all container-based applications operating on the OpenShift platform.

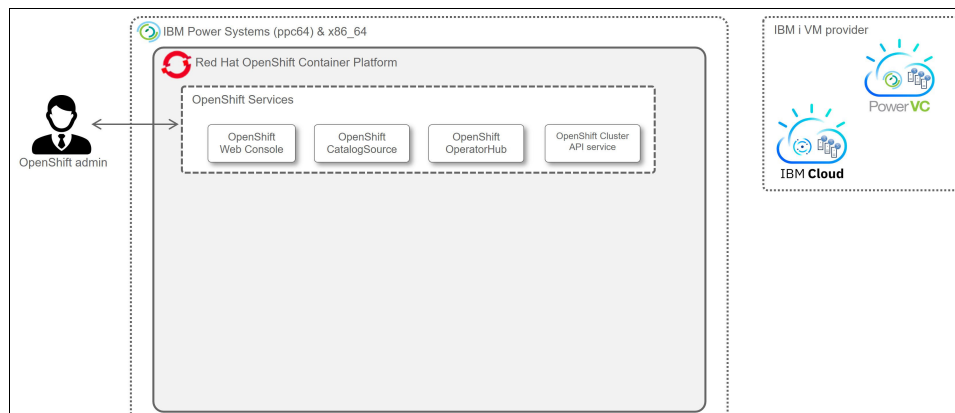


Figure A-15 OpenShift administrator role in Merlin integration

This role encompasses the following operations related to Merlin:

- ▶ Utilizing OpenShift CLI or GUI console to gain access.
- ▶ Executing the initial installation and deployment of the Merlin Platform.
- ▶ Reviewing and managing objects, configurations, and settings specific to Merlin.
- ▶ Conducting platform upgrades for Merlin.
- ▶ Gathering various tiers of logs for IBM service evaluation.

Note: The OpenShift environment can be situated on an IBM Power Systems server or any location compatible with current OpenShift implementations. Additionally, OpenShift can be hosted within a cloud instance, such as IBM Cloud (IBM Power Virtual Servers), or within any cloud platform that accommodates OpenShift environments. Clients who already have workloads functioning in the cloud can effortlessly extend their operations by integrating Merlin into an OpenShift environment within the cloud.

Merlin is specifically designed for Red Hat OpenShift containers, applicable to both IBM Power Systems (ppc64) and x86 architectures.

Merlin platform in the Merlin architecture

The next individuals in the spotlight are the Merlin administrators, tasked with ensuring the efficient integration and connectivity of all applications within the Merlin platform. Their responsibilities encompass setting up the appropriate Git repositories and establishing the necessary connections between components. A foundational understanding of concepts such as Git and Jenkins aids these administrators in directing resources to the correct libraries and repositories.

Note: While some familiarity with Git and Jenkins is essential, in-depth knowledge of Linux or OpenShift is not a prerequisite for the role.

It is worth mentioning that in smaller environments, these responsibilities can be undertaken by a single individual who possesses the skills to initiate OpenShift, facilitate its installation, and configure Merlin for operational use.

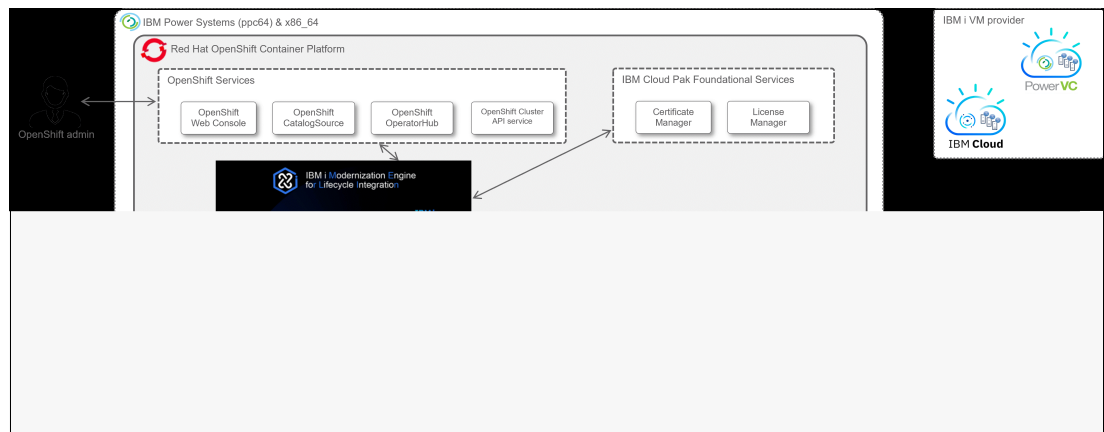


Figure A-16 Role and responsibilities of Merlin administrator

In the depicted scenario, a Merlin administrator assumes a pivotal role, wielding direct access to the Merlin platform GUI while orchestrating a spectrum of activities tailored for IBM i users. The scope of responsibilities entails crucial operations to ensure the platform's efficient functionality for IBM i users. These include, but are not limited to:

- ▶ The installation and deployment of Merlin tools, encompassing essential components such as the IDE, CI/CD functionalities, and more.

- ▶ Proficiently managing user accounts within the Merlin ecosystem.
- ▶ Ensuring the security and confidentiality of sensitive information entrusted to the platform.
- ▶ Exerting control over authorities and permissions, establishing a well-defined hierarchy of access.
- ▶ Diligently monitoring resource consumption and utilization, optimizing the platform's efficiency.
- ▶ Collaborating closely with OpenShift administrators to promptly address and resolve potential issues, nurturing a synergistic partnership for the platform's overall integrity and reliability.

Merlin platform and tools architecture

In the context of the Merlin platform and tools architecture, a distinct focus emerges. The objective is to empower developers, enabling them to engage in their development tasks using intuitive browser-based interfaces. This anticipated progression involves a certain learning curve, vital for mastering the installation of these components, whether within a Linux LPAR or across a suite of Linux partitions, contingent on the operational flow. Beyond this initiation, the subsequent journey is characterized by the accessibility of user-friendly browser-driven functionalities. Importantly, these functionalities play a pivotal role in orchestrating fundamental CI/CD processes, thereby enhancing the efficiency of the development experience as illustrated in Figure A-17.

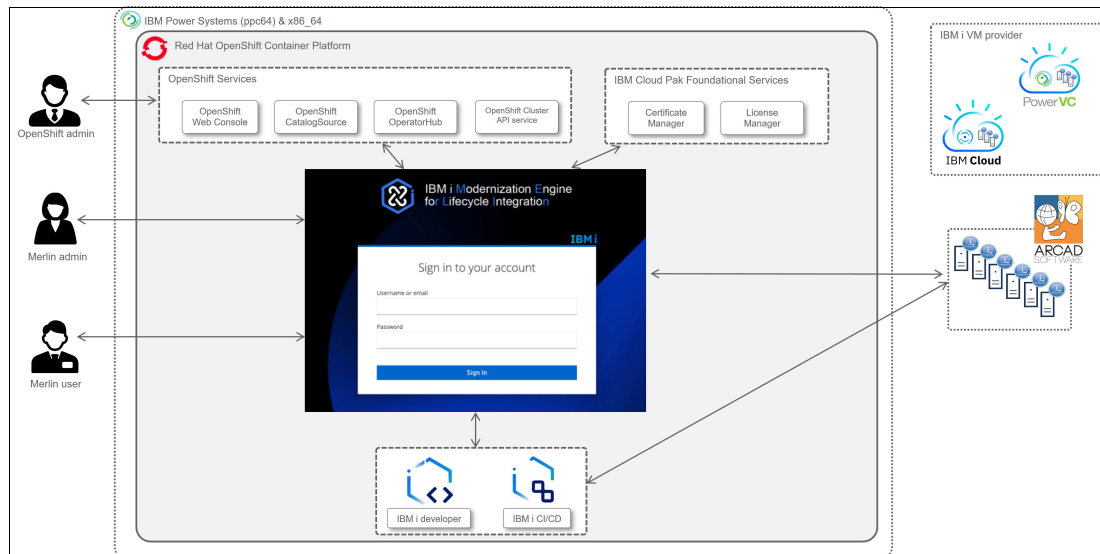


Figure A-17 Empowering developers with Merlin platform and tools

In the depicted scenario, a Merlin user enjoys direct access to the Merlin platform and tools. This user engages with Merlin to achieve the following objectives:

- ▶ Access and utilize the deployed IDE, CI/CD, and other functionalities.
- ▶ Ably manage inventory, user profiles, and credentials for targeted systems.
- ▶ Independently oversee the lifecycle of Merlin tools, subject to the requisite authority.
- ▶ Skillfully create Restful services on designated IBM i systems.

Merlin tools for IBM i CI/CD

Drawing on ARCAD's expertise in IBM i DevOps, our approach to deployment offers versatility to suit different scenarios. You have the option to utilize your existing Jenkins server or make use of the out-of-the-box setup, where we establish a Jenkins server for you and integrate it effectively with ARCAD components. This integration simplifies your CI/CD

workflow and enhances efficiency. Importantly, Merlin provides a user-friendly GUI tailored specifically for IBM i, ensuring a smooth experience. Refer to Figure A-18 for a visual representation of these capabilities.

Key features:

- ▶ Simplified Jenkins complexity: Merlin shields you from the intricacies of Jenkins, enabling you to concentrate on your IBM i CI/CD processes.
- ▶ Choice of deployment: You can choose to deploy without a Jenkins server, utilizing your own Jenkins instance, or take advantage of the provided Jenkins server integrated with ARCAD plugins.
- ▶ Flexible profile management: Private and public profiles offer a robust means to generate Jenkins pipelines dynamically. These profiles can be easily shared among Merlin users, promoting collaboration and consistent practices

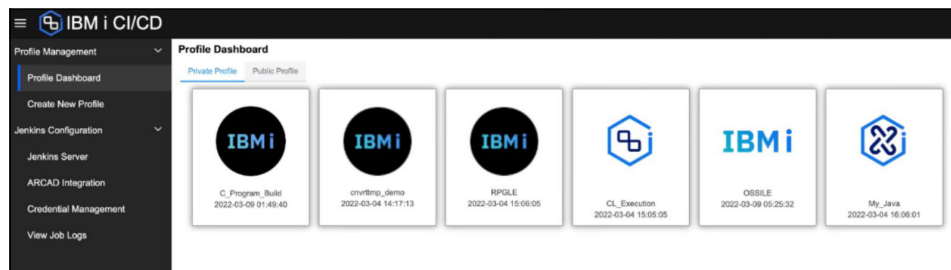


Figure A-18 IBM i CI/CD GUI with Merlin tools

A.1.9 Merlin for IBM i developers

In the context of code development, a natural development environment is crucial. Historically, IBM i featured PDM on the 5250, a facet ingrained in its development landscape. However, its applicability in the contemporary landscape is less normalized. The Rational Developer product, a well-regarded solution, remains effective for numerous users and will continue to do so in the foreseeable future.

In the next Figure A-19, you can observe the representation of the many developers who are still utilizing PDM.

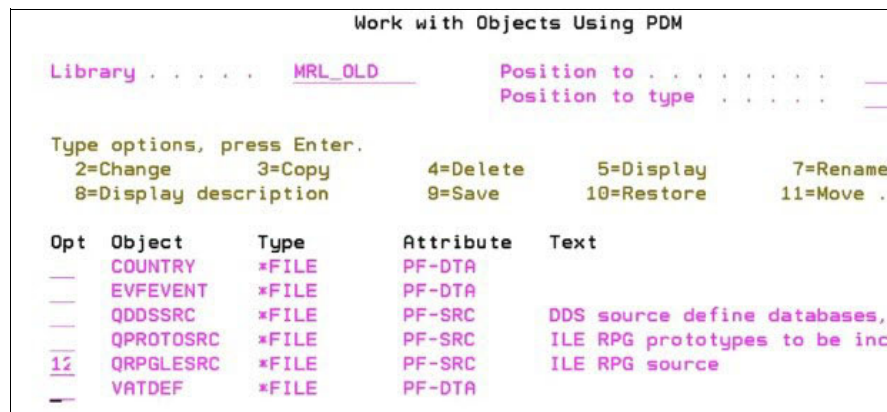


Figure A-19 Work with objects using PDM on 5250 interface

Yet, the primary aim is to assist IBM i customers seeking to adopt Git as their source control repository while embracing a modern browser-based development arena. Thus, IBM crafted a code-ready workspace, harnessing Eclipse Theia and Che along with an array of code

plugins. These components converge to deliver genuine code comprehension, comprehensive formatting, and a deep understanding of languages such as RPG, COBOL, and other native ILE types.

What adds significance is the integration of ARCAD's longstanding tools that have played a pivotal role in development pursuits over the years. Features such as effortless RPG conversion to modern free format and immediate access to an impact analysis tool are now essential aspects rather than mere afterthoughts within this innovative framework.

Figure A-20 demonstrates the IDE, setting up your development environment, the incorporation of rich editing capabilities, and building and compiling your project. Designed for contemporary developers, the offering includes natural Git integration, intelligent Build functions, self-contained projects, code comprehension, and integrated impact analysis.

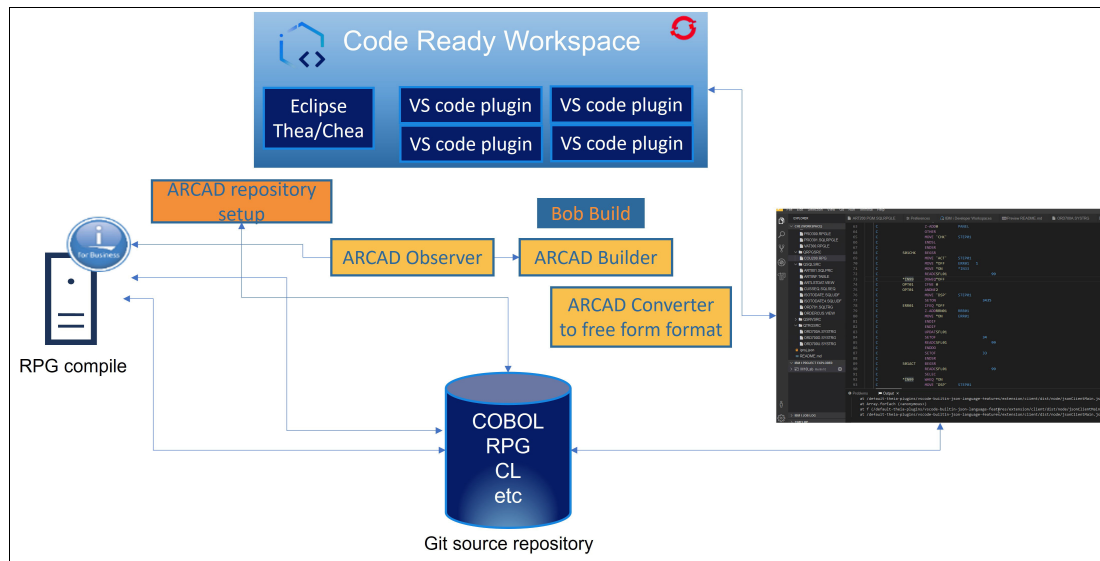


Figure A-20 Comprehensive development workflow with integrated tools

Note: BOB refers to the “Build on the Board” feature. BOB is a tool provided by ARCAD that facilitates the automated compilation and build process for RPG and COBOL applications on the IBM i platform. It allows developers to initiate the build process directly from a visual board or interface, helping to streamline the development workflow and enhance efficiency. The BOB tool is designed to integrate with DevOps practices and continuous integration processes, allowing for faster and more automated application builds.

Coexisting advancements: RDi and Merlin in IBM i development

Merlin introduces a fresh approach to code development and modernization, coexisting alongside Rational Developer for i. Rather than replacing RDi, Merlin provides an alternative option for developers to choose between workstation-based development with RDi or the browser-based, container-oriented environment of Merlin. Both solutions hold significant value within the IBM i development community, and IBM remains committed to enhancing and supporting both platforms.

While RDi caters to creating and updating native ILE applications on IBM i, Merlin brings a holistic Continuous Integration/Continuous Deployment (CI/CD) ecosystem centered around Jenkins. It serves as more than just an Integrated Development Environment (IDE), offering a comprehensive suite of tools and plugins to facilitate modern development practices. Merlin

equips developers with features such as Fixed to Free conversion, integration with Git-based source control, and real-time application impact analysis. Furthermore, it integrates with automated build and deployment pipelines, streamlining the entire development lifecycle.

A key distinction lies in the modernization capabilities of Merlin. Code crafted within Merlin can still be modified using RDi. While SEU (Source Entry Utility) can also be used for further code modification, Merlin's support for the latest RPG versions emphasizes a shift towards contemporary coding approaches, encouraging developers to permit newer paradigms for enhanced efficiency and sustainability.

IBM Merlin - ARCAD - powered tooling for enhanced development

Merlin has undergone significant enhancements achieved by integrating existing products and introducing novel elements. One example of this evolution is the builder facet, now enriched with a new web server and command-line interfaces. This integration covers various archive tools bundled with the core Merlin product. At the heart of this integration lies the Metadata repository, prominently featured in the lower right corner of the screen shown in Figure A-21. This repository is important, facilitating shared access for critical components such as Builder, Transformer RPG, and Observer – integral parts of the Merlin ecosystem. Its purpose is to connect these products, enabling real-time awareness of any changes or additions to objects.

This robust integration extends its benefits to both the Integrated Development Environment (IDE) and the Continuous Integration/Continuous Delivery (CI/CD) process. Users can initiate actions such as impact analysis directly from the IDE, while in the CI/CD pipeline, objects are automatically built along with their dependencies, all thanks to the consistently maintained metadata repository. This automated process eliminates the necessity of manually managing make files, a task that can become unwieldy in enterprise-level settings. Importantly, the Transformer RPG component serves as the initial step in the modernization journey. It enables transitioning code to a contemporary standard – fully free-format RPG – before engaging the broader range of modern tools and coding capabilities offered by Merlin.

This intricate integration and comprehensive enhancement underscore Merlin's pivotal role as a potent modernization engine, focused on lifecycle integration.

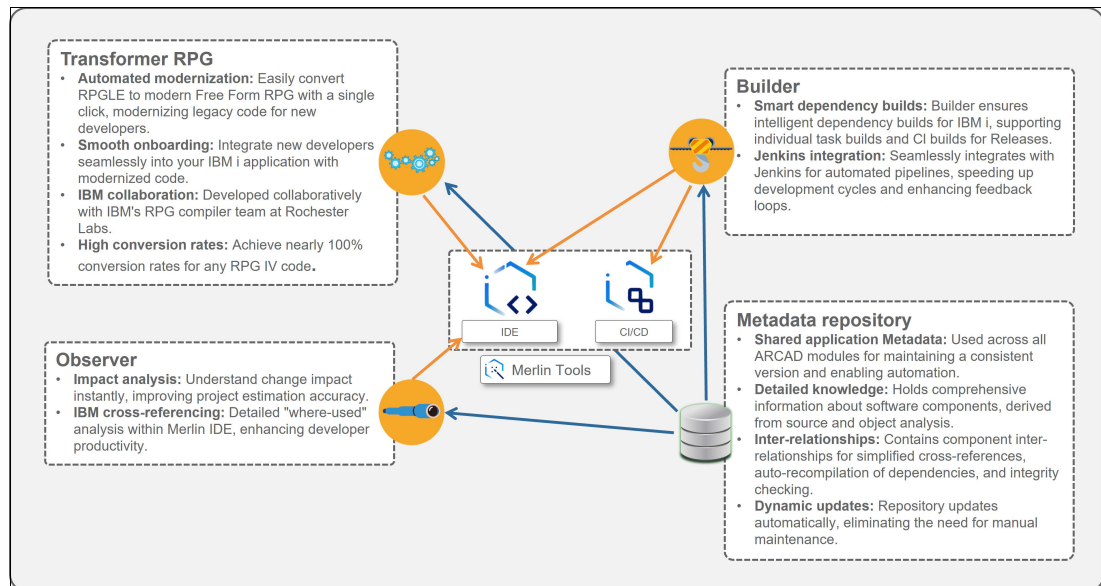


Figure A-21 IBM Merlin - ARCAD - powered tooling for enhanced development

Developer Merlin

The Developer Merlin environment provides a range of capabilities to enhance the development process:

- ▶ **Connections:** Includes features such as inventory management, credentials management, and template setup, enabling efficient access to the resources needed for development tasks.
- ▶ **Tools:** Developer Merlin offers a suite of tools, including those that have been deployed and configured for specific tasks. Of particular importance is the IBM i Developer tool, which empowers developers with seamless integration. Within this tool, you can perform actions such as right-clicking to run applications, streamlining the development workflow.
- ▶ **Create workspace:** This function allows developers to establish a dedicated workspace tailored to their requirements, ensuring an organized and efficient development experience.

Refer to the Figure A-22 for an illustrative portrayal of the initial workspace within IBM i - Developer, providing a visual representation of the starting point for developers using Merlin.

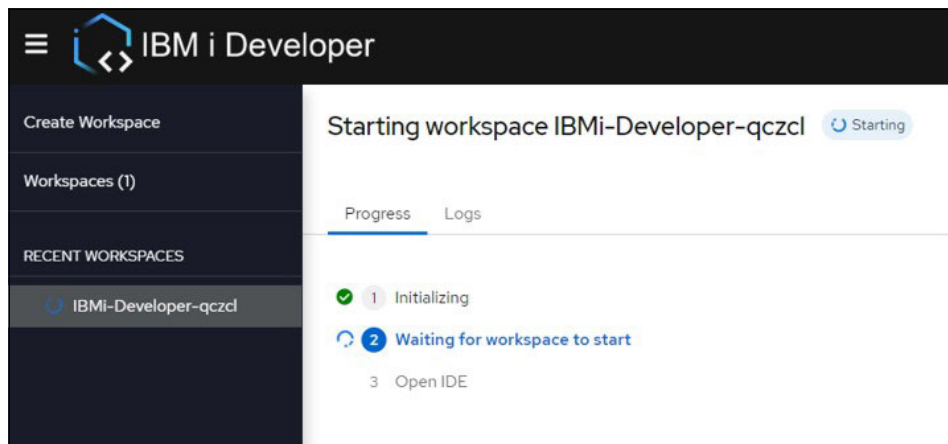


Figure A-22 Starting workspace in IBM i - Developer

Development Flow

Below are the key principles that define the Merlin development flow:

- ▶ **Inspired by GitFlow:** The development flow model takes inspiration from the GitFlow methodology, a well-established branching strategy. This approach provides a structured framework for managing code changes, releases, and collaboration among developers.
- ▶ **Adaptable:** The development flow is designed to be adaptable, accommodating various project requirements and team dynamics. It can be tailored to suit the specific needs of the development team, making it versatile and flexible.
- ▶ **Master and development - no direct changes:** The primary development branches, namely 'Master' and 'Development,' are kept free from direct changes. Instead, developers work on feature branches or other specialized branches, ensuring that the main development branches remain stable and reliable.
- ▶ **Other Branches:** The development flow includes several types of branches that serve distinct purposes:
 - Feature branches are created for developing new features or functionalities. These branches allow developers to work on isolated changes without affecting the main codebase.

- Release branches are used to prepare the codebase for a new release. They are ideal for bug fixes, last-minute adjustments, and testing before a release.
- HotFix branches are created to address critical issues in the production environment. They enable swift fixes without interrupting ongoing development efforts.
- ▶ Branch – ARCAD version: Each branch created within the development flow is accompanied by an associated ARCAD version. This version management ensures proper tracking and integration of changes, providing clear visibility into the status and progress of development activities.

Note: Git, a distributed version control system, offers several compelling advantages for source code management:

- ▶ Line-level visibility of changes: Unlike traditional change management systems, Git provides a granular view of changes at the line level. This allows developers to precisely track modifications, enhancing transparency.
- ▶ Enhanced management of concurrent development: Git's decentralized nature allows multiple developers to work on different branches simultaneously, facilitating smoother collaboration and concurrent development efforts.
- ▶ Explicit merges: When two changes are merged into the same codebase, Git makes this process explicit. This ensures that changes are intentionally combined, reducing the risk of accidental conflicts.
- ▶ Controlled commits: Git's commit process includes conflict checks, enabling developers to review and manage potential conflicts before finalizing changes. This enhances code quality and reduces integration challenges.
- ▶ Offline usage: Git's offline capabilities enable developers to track local changes even when disconnected from a network. This flexibility supports productivity in various work environments.
- ▶ Incredible traceability: Git's version control offers unparalleled traceability, allowing you to track the history of changes, contributors, and decisions made throughout the development lifecycle.

Managing IBM i source with Git and ARCAD

Effectively managing your IBM i source code is crucial for a development process. By combining the power of Git version control and ARCAD's capabilities, you can optimize your source management workflow. The following steps outline the process of handling IBM i source code within this collaborative environment:

- ▶ Create empty Git repository:
 - Begin by establishing an empty Git repository to serve as the foundation for your source code management.
- ▶ Configure application in ARCAD:
 - Configure your application within ARCAD, specifying components such as *Mxx_DTA*, *Mxx_OBJ*, and *Mxx_SRC*. Link these components to the Git repository for seamless integration.
- ▶ Generate source for objects:
 - Utilize the **GENSCMSRC** command to generate source code specifically for objects. This process facilitates the progression of source code transition into your infrastructure.

► Load Git repository with source:

Employ the **LODSCMREP** command to populate the Git repository with the generated source code. This step ensures that your code is effectively managed within the version control system, enhancing collaboration and traceability.

Merlin preferences

Within Merlin, user preferences are meticulously designed to enhance the development experience. These preferences are tailored to integrate with ARCAD's tools, ensuring a cohesive workflow.

1. Builder:

- Port: 5252: This preference configures the communication port for Builder, ensuring interaction between components.

2. IBM i Developer:

- Build settings: The preferences for IBM i Developer encompass a range of build settings. These include options related to Build on Build (BOB).
- Formatting options are available, allowing developers to tailor their development environment to their coding style and preferences.

3. Customizable color scheme: One of the user-friendly preferences provided by Merlin is the ability to modify the color scheme. This customization feature empowers developers to create a coding environment that is visually pleasing and conducive to their individual needs.

Git integration in Merlin

Integrating Git functionality within the Merlin workspace brings enhanced efficiency and collaboration to your development process. The integration of **git** command allow for effective version control and access to various collaborative tools. The following points outline the key steps and benefits of this integration:

1. Within the Merlin workspace, you can integrate Git functionality for version control and collaborative development.
2. Utilize the F1 key to access a comprehensive list of Git commands, enabling efficient navigation and execution.
3. Initiate the process by typing **git clone** within the command interface of Merlin's workspace. For a visual representation of this process, refer to Figure A-23 where the **git clone** command is demonstrated.

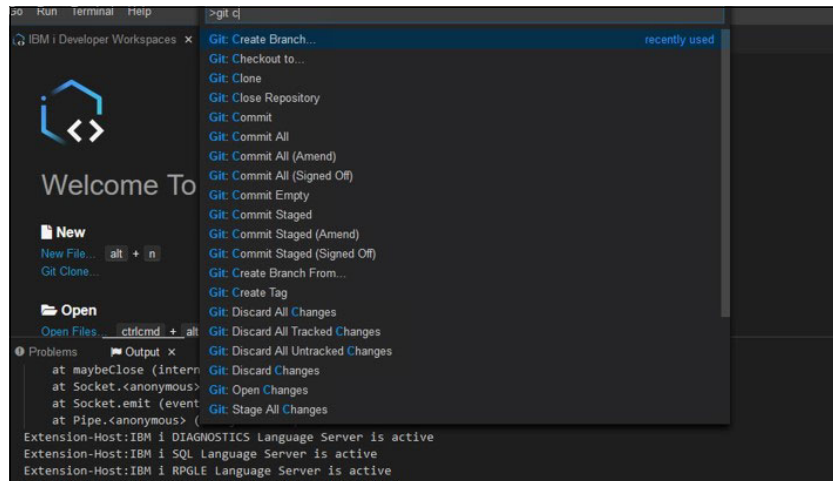


Figure A-23 Git clone command in Merlin workspace

4. Provide the SSH URL of the Git repository you intend to clone, establishing the connection between Merlin and the Git repository.
5. Upon the successful completion of the clone process, your source code becomes visible and accessible within the Merlin workspace. This integration streamlines version control and source code management, enhancing your development workflow.
6. To begin the process of creating a branch, locate the “Feature/xxxx” section, where the mapping between Git and ARCAD, labeled as “awrkvertyp,” is defined. As shown in Figure A-24.

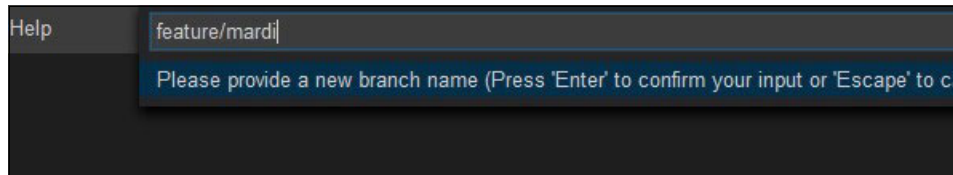


Figure A-24 Creating a branch in Merlin: Mapping Git and ARCAD

7. To create the branch, press F1 and select “git create branch”. As shown in Figure A-25.

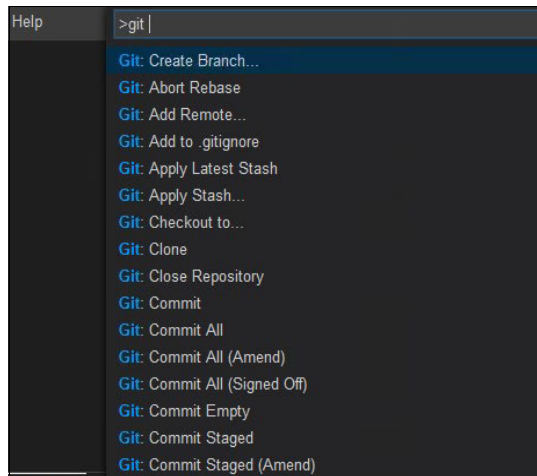


Figure A-25 Creating a new branch in Merlin

8. Further, in the bottom left corner, click on “master” to proceed with the branch creation process. Figure A-26 is displayed.

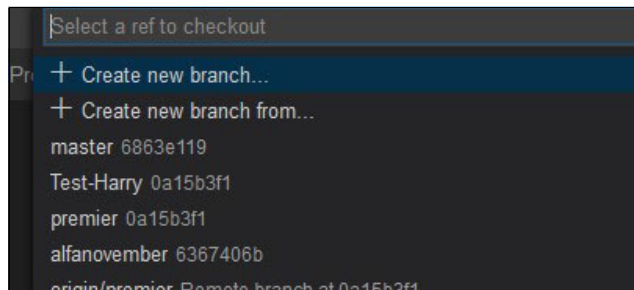


Figure A-26 Selecting 'master' to begin branch creation

9. After making changes to your local repository, use the **push** command to upload your committed changes to the remote Git repository. This synchronizes the changes you made on your local machine with the online repository, ensuring that other team members can access your updates.

10. As a result of pushing your changes, the remote Git repository is updated with the latest changes you committed. This allows other team members to access and work with the most recent version of the codebase.
11. A webhook is a mechanism that allows real-time communication between different systems. In the context of Git and ARCAD Builder, you can set up a webhook to notify Builder about certain events in the remote repository. This integration is enabled by copying the GitHub webhook from Builder's webhook processing tool, known as “smeem.” To use this feature, ensure that webhook processing is activated in Builder, and use the provided webhook link (for example, <https://smeem.io/IzfhozWff1rfG10t>) to establish the connection as shown in Figure A-27

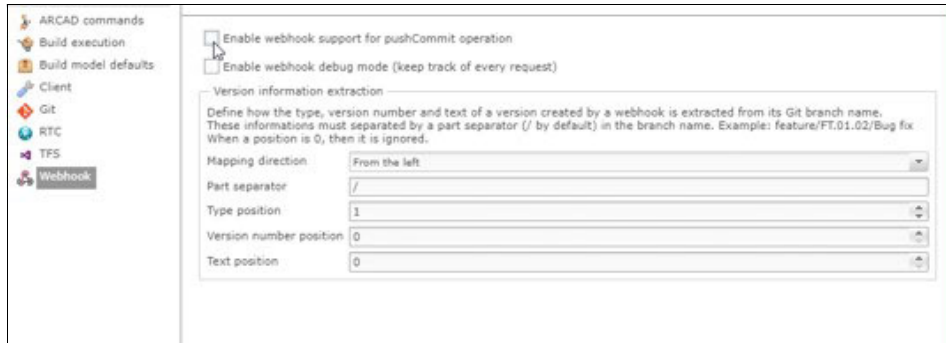


Figure A-27 Webhook integration between Git and ARCAD Builder

12. Upon using the **check version** command, an automatic commit is generated in your local repository. By pulling from your local repository, you retrieve the latest commit message from the remote repository. This process ensures that you are always working with the most up-to-date code and information, promoting collaboration and reducing potential conflicts.
13. To incorporate the necessary IBM i views, right-click on CHE, as illustrated in Figure A-28

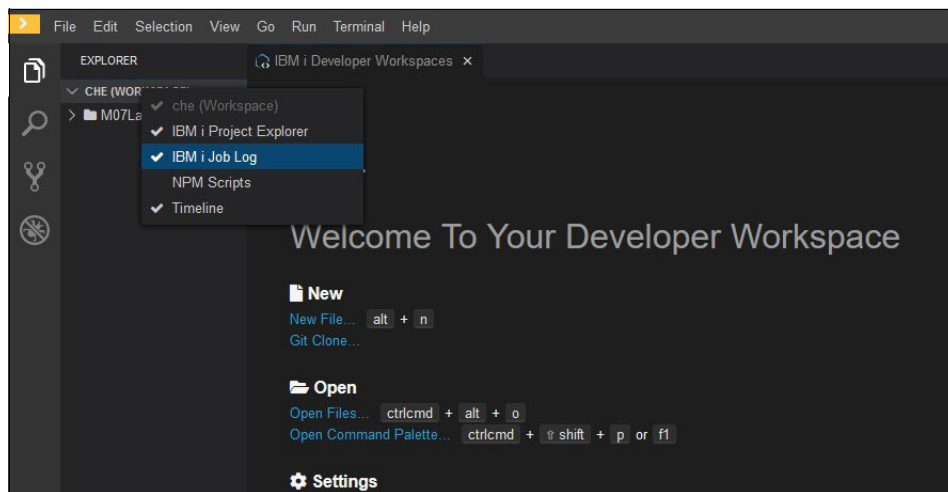


Figure A-28 IBM i views integration in CHE

IBM i project explorer

IBM i project explorer offers essential features for integration and efficient development within the IBM i environment:

- ▶ **Variables:** Manage and track variables used in your development projects, ensuring accurate data handling and processing.

- ▶ Library lists: Easily configure library lists, allowing you to access the necessary libraries and resources for your projects without the hassle of manual setup.
- ▶ Object libraries: Access and organize object libraries efficiently, simplifying the management of your IBM i resources.
- ▶ My queries: Utilize built-in query functionalities to retrieve specific information from your IBM i system, enhancing your ability to gather relevant data for your projects.

ARCAD view

The ARCAD view is a user interface provided by the ARCAD software suite that offers a consolidated and organized perspective into various aspects of the software development lifecycle. You can efficiently manage your development tasks and processes:

- ▶ Sites: Gain a comprehensive overview of your different development environments or locations, allowing you to organize and navigate your projects effectively.
- ▶ Builds: Track the progress of builds, ensuring a clear understanding of the current status of your development efforts.
- ▶ Versions: Easily access and manage different versions of your projects. Each version is linked to a specific branch, providing a way to navigate between various stages of development.

Prompting

Prompting in the context of Merlin refers to the interactive assistance provided to developers during various stages of application development. It offers guidance and suggestions as developers write code, aiding in the creation of accurate and efficient programs. Prompting enhances the development experience by reducing errors, improving consistency, and increasing productivity.

Changed source

Changed source in the Merlin platform is tracked through a “Modified” flag, which indicates that alterations have been made to the code. These changes are managed within the platform's source control system.

Git compare

Merlin offers a Git compare feature that allows developers to efficiently analyze differences between various versions of source code. This tool enhances collaboration by providing an intuitive visual representation of changes, aiding in code review, error identification, and maintaining code quality throughout the development lifecycle.

Git process

As part of the Git process within Merlin, developers can conveniently stage changes and commit them locally. This ensures that modifications are organized and tracked effectively before they are pushed to the shared repository, contributing to a structured and controlled development workflow.

Central Git repository

In Merlin, the central Git repository serves as a central hub for collaborative development. When developers push their changes to a specific branch in this repository, the repository is updated with the latest modifications, promoting efficient collaboration and version control within the development team.

Changes received by Git

When changes are committed and pushed to the central Git repository, these modifications are received by Git, ensuring that the latest updates are accessible to all team members collaborating on the project.

ILE RPG offers extensive support

ILE RPG, with its extensive support, offers a range of features for enhanced development:

- ▶ Outline: Provides a clear overview of the code structure.
- ▶ Model creation: Simplifies the representation of the code's logic.
- ▶ Easy navigation: Streamlines moving between different sections of the code.
- ▶ Hover information: Displays contextual information when hovering over elements.
- ▶ Procedure definition information: Presents details from procedure definitions.
- ▶ Procedure call analysis: Right-clicking allows you to explore references and usages of procedures, providing a deeper understanding of their impact.
- ▶ Collapsible code blocks: Allows you to collapse sections of code for improved readability and focus.

Tokenization

Tokenization in the context of Merlin refers to the process of categorizing and highlighting different elements in your code using appropriate colors. This visual differentiation helps you quickly identify and distinguish between various components within your codebase, leading to improved readability and ease of understanding.

Code Formatting

Merlin's Code Formatting ensures consistent and organized code for readability. Customize preferences for automatic formatting alignment with your style. Right-click selected code and choose "Reformat" to instantly apply chosen rules. This maintains uniformity and enhances comprehension.

Refactoring

Merlin's Refactoring enhances code structure and readability without sacrificing functionality. Rename symbols intelligently, updating code and model for consistency. "Shift+Enter" previews changes before committing. The model auto-updates, reflecting modifications. This approach ensures accurate and efficient code improvement.

Content Assist

Content Assist in Merlin provides intelligent suggestions as you code. Use "Ctrl+Space" to invoke it. It works for both language and model, enhancing accuracy and speed. The live problem view identifies and highlights issues, allowing direct navigation and automatic updates as you fix them.

SQL

SQL in Merlin brings advanced features for efficient database interaction and management.

- ▶ Tokenization: Clearly divides SQL into meaningful elements for easy comprehension and editing.
- ▶ Formatting: Ensures consistent and readable SQL code by automatically applying formatting rules.
- ▶ Code collapse: Organizes SQL blocks, making it simpler to navigate and focus on relevant sections.
- ▶ Embedded SQL: Integrate SQL statements within host languages, enhancing database interaction within application code.

ARCAD Transformer RPG

ARCAD Transformer RPG is a powerful tool that facilitates the modernization of RPG code, enabling it to adapt to contemporary coding standards and practices. However, there are certain aspects that the transformation process does not consider:

- ▶ Specifications not available in Free Form: Traditional I and O specifications are not available in Free Form RPG.
- ▶ Not managed in F / D specs: Certain elements such as primary files (P), secondary files (S), table files (T), or address files are not managed in Free Form RPG. Additionally, D-specs with FROMFILE / TOFILE clauses are excluded.
- ▶ Unconverted operations: Certain RPG operations such as MHHZO, MHLZO, MLHZO, MLLZO are not automatically converted during the transformation process.

Furthermore, there are specific cases where operation codes cannot be converted as follows:

- ▶ TIME: If the result field length is equal to 14 characters.
- ▶ SCAN, CHECK, CHECKR: When the result field is an array.
- ▶ BITON, BITOFF: When factor 2 is a named constant.
- ▶ POST: When the result field (data structure name) is used.
- ▶ MOVE, MOVEL: When the factor 2/result is a varying-length field.
- ▶ MOVEA: When the field is defined as a CONST parameter.
- ▶ KLIST, KFLD: When located or used in a COPY clause.
- ▶ CALL, PARM: For CALL operations, the indicator “LR” (positions 75-76) and the CALL Pgm(idx) syntax are not converted.
- ▶ GOTO, TAG: GOTO within a sub-procedure and TAG in the “Main” program, or when GOTO and TAG do not comply with structured programming. Also, when TAG is used by WHENEVER GOTO (SQL).

A.1.10 Merlin requirements

The installation of IBM i Modernization Engine for Lifecycle Integration is flexible, allowing you to install it using either the OpenShift web console or the CLI (`oc` command). This versatility provides you with options for an easy and convenient installation process tailored to your preferred method.

Versions of the product that are compatible and officially endorsed

Table A-4 provides a comprehensive overview of the versions that are both compatible and officially endorsed for the installation of Merlin. By ensuring that the product versions align harmoniously, IBM i and OpenShift Container Platform users can confidently embark on the modernization journey facilitated by Merlin. The following table presents a clear snapshot of the supported versions, allowing you to make informed decisions that pave the way for a successful implementation of Merlin within your IT environment.

Table A-4 Supported versions of IBM i and OpenShift Container Platform for Merlin installation

IBM i	OpenShift Container Platform
7.3	4.8
7.4	4.9
7.5	4.10

Necessary OpenShift resources

Table A-5 outlines the crucial CPU requests, CPU limits, memory requests, and memory limits that play a pivotal role in ensuring the optimal functioning of Merlin within the OpenShift environment. This adherence to resource allocation guidelines is integral to the successful implementation of the IBM i Modernization Engine.

Table A-5

Name	CPU request	CPU limit	Memory request	Memory limit	Note
Merlin	2.5 ^a	5 ^b	7G ^a	15G ^b	
IBM i Developer Tool	0.5 ^a	2.7 ^b	1.5G ^a	3G ^b	The resource is per each instance ^c
IBM i CI/CD	0.5 ^a	1 ^b	1G ^a	2G ^b	The resource is per each instance ^c

a. Request signifies the minimum required amount.

b. Limit signifies the maximum anticipated utilization.

c. When an administrator installs either of IBM i Developer or IBM i CI/CD Tools within an OpenShift project, it corresponds to one instance.

IBM i requirements

Prerequisites for the IBM i environment encompass the following:

- ▶ IBM i 7.3 or a more contemporary release, complemented by the latest application of the HTTP PTF Group.
- ▶ Rational Development Studio (5770-WDS) is an essential requirement for the compilers, enabling the conversion of source code into object code.

Entitlement

Clients can acquire Merlin through [IBM Passport Advantage](#), enabling a smooth and efficient acquisition process. Upon purchase, users authenticate via their IBM ID on Passport Advantage. Subsequently, a designated entitlement key associated with the acquired product is activated within the IBM Marketplace. This entitlement key, coupled with an active paid entitlement, grants customers access to the container images available within the Entitled Registry.

Price: Merlin follows a “per-developer” pricing model, aligning with its deployment within the Red Hat OpenShift Container Platform (OCP). Utilizing the inherent license monitoring mechanism of OCP, Merlin employs the VPC (Virtual Processor Core) framework. To secure Merlin entitlement, customers can place an order for 1 VPC unit per developer, resulting in the creation of an individual CodeReady workspace for each developer. This offering is available at a rate of \$4500.00 per VPC.

Installing IBM i Merlin in air-gapped environment

For detailed step-by-step guidance on installing IBM i Merlin in an air-gapped environment, please refer to the comprehensive instructions provided at: [Install IBM i Merlin in Air-Gap environment](#). This resource covers essential prerequisites, the setup of a bastion host, configuration of the local Docker registry, installation procedures for IBM i Merlin, along with comprehensive guidance on mirroring images and configuring the cluster. Additionally, you can find clear instructions for creating the IBM i Modernization Engine for Lifecycle Integration catalog source. Following these detailed steps ensure a smooth and successful installation process.

Tip: For further information about installation Merlin and more, see [Installing Merlin](#)

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- ▶ *Introduction to IBM PowerVM*, SG24-8535
- ▶ *Deploying SAP Software in Red Hat OpenShift on IBM Power Systems*, REDP-5619
- ▶ *IBM Power Systems Cloud Security Guide: Protect IT Infrastructure In All Layers*, REDP-5659
- ▶ *Oracle on IBM Power Systems*, SG24-8485

You can search for, view, download or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

To determine the spine width of a book, you divide the paper PPI into the number of pages in the book. An example is a 250 page book using Plainfield opaque 50# smooth which has a PPI of 526. Divided 250 by 526 which equals a spine width of .4752". In this case, you would use the .5" spine. Now select the Spine width for the book and hide the others: **Special>Conditional Text>Show/Hide>SpineSize(->Hide)>Set** . Move the changed Conditional text settings to all files in your book by opening the book file with the spine:fm still open and **File>Import>Formats** the Conditional Text Settings (ONLY!) to the book files.

Draft Document for Review August 29, 2024 3:53 pm

8551 spine.fm 371



Using Ansible for Automation in IBM Power Environments

SG24-8551-00

ISBN DocISBN



(1.5" spine)

1.5" <-> 1.998"

789 <-> 1051 pages



Using Ansible for Automation in IBM Power Environments

SG24-8551-00

ISBN DocISBN



(1.0" spine)

0.875" <-> 1.498"

460 <-> 788 pages

Redbooks

Using Ansible for Automation in IBM Power Environments

SG24-8551-00

ISBN DocISBN



(0.5" spine)

0.475" <-> 0.873"

250 <-> 459 pages

Redbooks

Using Ansible for Automation in IBM Power Environments

(0.2" spine)

0.17" <-> 0.473"

90 <-> 249 pages

(0.1" spine)
0.1" <-> 0.169"
53 <-> 89 pages

To determine the spine width of a book, you divide the paper PPI into the number of pages in the book. An example is a 250 page book using Plainfield opaque 50# smooth which has a PPI of 526. Divided 250 by 526 which equals a spine width of .4752". In this case, you would use the .5" spine. Now select the Spine width for the book and hide the others: **Special>Conditional Text>Show/Hide>SpineSize(->Hide)>Set** . Move the changed Conditional text settings to all files in your book by opening the book file with the spine:fm still open and **File>Import>Formats** the Conditional Text Settings (ONLY!) to the book files.

Draft Document for Review August 29, 2024 3:53 pm

8551 spine.fm 372



Using Ansible for Automation in IBM Power

SG24-8551-00

ISBN DocISBN

(2.5" spine)
2.5" <-> mmm.n"
1315 <-> mmm pages



Using Ansible for Automation in IBM Power Environments

SG24-8551-00

ISBN DocISBN

(2.0" spine)
2.0" <-> 2.498"
1052 <-> 1314 pages





SG24-8551-00

ISBN DocISBN

Printed in U.S.A.

Get connected

