Ansible for Kubernetes by Example

Automate Your Kubernetes Cluster with Ansible

—

Luca Berton

Apress®

Ansible for Kubernetes by Example

Automate Your Kubernetes Cluster with Ansible

Luca Berton

Apress®

Ansible for Kubernetes by Example: Automate Your Kubernetes Cluster with Ansible

Luca Berton
BOURNEMOUTH, UK

ISBN-13 (pbk): 978-1-4842-9284-6 ISBN-13 (electronic): 978-1-4842-9285-3
https://doi.org/10.1007/978-1-4842-9285-3

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Divya Modi
Development Editor: James Markham
Coordinating Editor: Divya Modi
Copy Editor: Kezia Endsley

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 New York Plaza, Suite 4600, New York, NY 10004-1562, USA. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub (github.com/Apress/Ansible-for-Kubernetes-by-Example-by-Luca-Berton). For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

For my son Filippo, the joy of my life.

Table of Contents

About the Author xi

About the Technical Reviewer xiii

Acknowledgments xv

Introduction xvii

Chapter 1: Modern IT Infrastructure and Hello App 1

Modern IT Infrastructure (DevOps and IaC) 1

The Move to Containers 4

Ansible by Red Hat 7

The Cloud Native Computing Foundation 8

Kubernetes Support 9

Kubernetes Distributions: OpenShift, Rancher, EKS, AKS, and GCP 9

Containers and Pods 11

Creating a Hello App 12

Linux Base Images 13

Enterprise Linux-Based Images 15

Container Security 17

The Hello Dockerfile 18

The Hello Application 19

Building the Hello App 22

Running Hello in Docker 22

Deploying Hello in Kubernetes 24

Deploying Hello in Operator 25

Key Takeaways 25

Chapter 2: Ansible Language Code 27
What Is Ansible? 27
Provisioning 28
Configuration Management 29
Application Deployment 29
Ansible Open-Source vs Commercial Options 29
Ansible's Architecture 30
UNIX Target Node 31
Windows Target Node 32
Ansible Installation 34
Getting Started with Ansible 36
Running Your First Ansible Ad Hoc Command 36
Creating a Basic Inventory 37
Ansible Code Language 37
Ansible Inventory 38
Ansible Playbook 40
Ansible Roles 51
Ansible Collection 54
Ansible Execution Environment 55
Key Takeaways 61

Chapter 3: Ansible for Containers 63
Ansible for Containers 63
Install Docker on Linux and Windows 63
Install Docker in Debian Linux 65
Install Docker in Red Hat Linux 67
Install Docker on Windows 70
Flatpak in Linux 71
Snap in Linux 74
Deploy a Web Server in a Container 77
Apache with Docker for Debian-like Systems 78
Apache with Podman for Red Hat-like Systems 80

Use Vagrant and Packer 82
Vagrant 83
Packer 84
Key Takeaways 85

Chapter 4: Ansible for K8s Tasks 87

Kubernetes Objects 90
Control Plane vs Data Plane 91
kubectl 92
GitOps Continuous Deployment 93
Jenkins 94
VMWare Tanzu Application Platform 95
Set Up Your Laboratory 95
Virtual Machines 96
Raspberry Pis 97
Kubespray 99
OpenShift Local 100
hetzner-ocp4 103
Create a Cluster with Minikube 104
Kubeadm 106
K3s Lightweight Kubernetes 107
Kubernetes Upgrade 107
Create a Cluster with kOps 109
Configure Ansible for Kubernetes 109
Ansible Troubleshooting 113
401 unauthorized 113
Kubernetes 114
OpenShift 116
x509 error 118
kubeconfig 118
Configure a Python Virtual Environment 120

Configure an Ansible Execution Environment 123
Create a Namespace 126
Report Namespaces 129
Report Deployments in Namespace 131
Create a Pod 136
Create a Secret 139
Use a Service to Expose Your App 141
Kubernetes Networking 143
Scale Your App 148
Auto-scaling 153
Update Your App 154
Assign Resources to Kubernetes K8s Pods 157
Metrics 158
CPU Resources 158
Memory Resources 161
Namespace Resources 162
GPU Resources 163
Configure a Pod to Use a Volume for Storage 163
Apply Multiple YAML Files at Once on Kubernetes 166
Key Takeaways 167

Chapter 5: Ansible for K8s Data Plane 169

Configuring a Java Microservice 171
The Demo Java Web Application 172
Stateless: Deploying PHP Guestbook Application with Redis 180
Kustomize: Do More with Less 184
Stateful: Deploying WordPress and MySQL with Persistent Volumes 186
Security Namespace (Pod Security Admission) 191
Security Pod Resources (AppArmor) 192
Security Pod Syscalls (seccomp) 193

Ansible Dynamic Inventory 194
Key Takeaways 199

Chapter 6: Ansible for K8s Management 201

The Helm Package Manager 202
Helm Repositories 204
Helm Packages 207
Helm Plugins 212
Deploy a Monitoring Tool 216
kube-prometheus 217
Ansible Collections 218
Helm Chart 219
Fetch Logs from Resources 220
Apply a JSON Patch Operation 222
Copy Files and Directories to and from a Pod 224
Manage Services on Kubernetes 225
Taint Nodes 227
Drain, Cordon, or Uncordon Nodes 229
Kubernetes Dynamic Inventory 232
Roll Back Deployments and DaemonSets 232
Set a New Size for a Deployment, ReplicaSet, Replication Controller, or Job 233
Security 233
AAA 235
OpenID Identity Provider 236
Calico 237
Key Takeaways 237

Chapter 7: Ansible for Kubernetes Cloud Providers 239

Cloud Architecture 241
Amazon Web Services (AWS) 244
Google Cloud Platform (GCP) 251

Microsoft Azure Cloud Services 255
Other Vendors 259
Key Takeaways 260

Chapter 8: Ansible for Enterprise 261

The Ansible Automation Platform 261
Event-Driven Ansible 268
IT Trends 272
Ansible Trusted Code 273
What's Next? 275
Thank You 276

Index 277

About the Author

Luca Berton is an Ansible automation expert who works for JPMorgan Chase & Co. and previously worked with the Red Hat Ansible Engineer Team for three years. He is the creator of the Ansible Pilot project. With more than 15 years of experience as a system administrator, he has strong expertise in infrastructure hardening and automation. An enthusiast of open-source, Luca supports the community by sharing his knowledge at different public access events. He is a geek by nature, Linux by choice, and Fedora, of course.

About the Technical Reviewer

Nikhil Jain is an Ansible expert with over 12 years of DevOps experience. He has been using Ansible and contributing to it from its inception. He currently works closely with Ansible engineering.

He is an open-source enthusiast and is part of the Ansible Pune Meetup Organizing team. He has presented multiple Ansible sessions at various global and local events. Apart from sitting in front of his computer automating things using Ansible, he loves watching sports and is a regular part of the local cricket team.

Acknowledgments

To my son, family, and friends, who make life worth living and whose support and encouragement make this work possible.

To all whom I've worked with over the years and shared any ideas for this book: Thank you for the knowledge you've shared.

Introduction

This book guides you through the process of automating a Kubernetes cluster using the Ansible open-source technology.

If you are an IT professional who wants a jargon-free explanation of the Ansible technology, this book is for you. This includes Kubernetes, Linux, and Windows systems administrators, DevOps professionals, thought leaders, and infrastructure-as-code enthusiasts, as well as information technology team members providing leadership to their businesses.

Infrastructure and operations (I&O) leaders look at the Ansible platform to implement Infrastructure as Code (IaC), Configuration as a Code (CaC), Policy as a Code (PaC), code pipelines, orchestration (K8s), DevSecOps, self-healing infrastructures, and event-driven automation methodologies.

This book is a powerful resource for computer engineers and leaders who believe that innovation, automation, and acceleration are the drivers of the booming business of tomorrow.

A successful infrastructure is a matter of improvements that are made little by little and good habits that you develop using automation during the journey.

In this book, you learn how to become more productive and effective using the Ansible open-source automation technology to deploy world-scale, cloud-native applications and services. Containers have revolutionized the IT industry, but it takes more than container runtime engines like Docker and Podman to handle these modern, huge workloads. This is where Kubernetes come in to play.

Kubernetes applications are designed as microservices, packaged in containers, and deployed in a hybrid cloud in multiple cloud platforms worldwide to serve your customers.

It's also possible to upgrade applications and services without impacting user performance in many ways: by rolling, canary, and blue/green environments.

The scriptable infrastructure of the cloud providers enables you to elastically meet the demands of traffic with a virtually unlimited compute powerhouse. Engineers have a great deal of power and responsibility for their business's success.

What Is In This Book?

This book provides in-depth content of the following topics:

- The Ansible code language for beginners and experienced users, with examples
- Ansible installation on the latest releases of Ansible
- Ansible for container management (Docker and Podman)
- Ansible for Kubernetes infrastructure examples
- Troubleshooting common errors

Development Environment

You do not need a specific IDE to benefit from this book. You simply need a base environment consisting of the following:

- A text editor: Terminal-based (VIM, Emacs, Nano, Pico, and so on) or GUI-based (VS Code, Atom, Geany, and so on)
- A workstation with the `ansible` or `ansible-core` packages installed
- Kubernetes or OpenShift cluster

Conventions Used in the Book

This is a practical book, with a lot of examples and terminal commands. There are also some Ansible language code samples.

The Ansible language code used in the Ansible Playbook examples mostly uses the YAML and INI formats.

Commands and code samples throughout the book are either inline (for example, `ansible [command]`) or in a code block (with or without line numbers), such as this:

```
---
# YAML file example
```

The Ansible language is written in the YAML format, which is a human-readable, data-serialization language that is extremely popular for configuration files. The code

follows the latest YAML 1.2.2 specification. It uses Python-style indentation and a more compact format for lists and dictionary statements. It's very close to JSON and can be used as an XML alternative. The YAML code was validated using the YAMLlint popular validator and tested in the most commonly used Ansible versions in the market.

The INI format used in Ansible inventories examples is a well-known format for configuration files since the MS-DOS operating system and uses key-value pairs for properties.

The terminal commands use the standard POSIX conventions and are ready for use in UNIX-like systems such as Linux, macOS, or BSD. Each command is assumed to be used by a standard user account when the prefix is the $ symbol (dollar) or by the root user when the prefix is the # symbol (number sign).

You are going to find this code in installation, code execution, and troubleshooting examples. The commands were tested in the most modern operating system versions available.

The Ansible code included in this book was tested by the author and the technical reviewer in a wide variety of modern systems and uses the Ansible best practices regarding playbooks, roles, and collections. It was verified using the latest release of the Ansible Linter.

Some code may intentionally break a specific Ansible best practice rule only to demonstrate a troubleshooting session and reproduce a fatal error in a specific use case.

Chapters at a Glance

This book aims to guide you through your journey of automating the Ansible platform for the Kubernetes (K8s) infrastructure. There are eight chapters, and the book is full of code samples and terminal commands to verify the results.

- Chapter 1 explores how modern IT infrastructure is organized inside an organization and outside to its customers and partners. You are going to learn about the history of Kubernetes, from the initial Google project to CNCF. You also learn about its most famous flavor: Red Hat OpenShift, Amazon EKS, and other public cloud providers. A practical example of the Hello App written in Python language is deployed as a container (Fedora, Ubuntu, Alpine Linux), built in Docker and running in Kubernetes.

- Chapter 2 dives deep into the Ansible language code. It explains the architecture, how to set up a target node, inventories, as well as playbooks, variables, filters, conditionals, handlers, loops, magic variables, vaults, templates, plugins, roles, and collections.
- Chapter 3 is all about containers—how to automate the installation of Docker in your operating systems, how to use the Flatpack and Snap technologies in Linux, and how to deploy the popular web server Apache as a containerized application in Debian/Ubuntu. To do that, you use Docker and Red Hat Enterprise Linux with the Podman runtime engine.
- Chapter 4 goes deep into Kubernetes, explaining the difference between the control plane and the data plane and how to use the kubectl utility. You learn how to set up your laboratory using virtual machines, Raspberry Pis, Minikube, Kubespray, Kubeadm, and OpenShift Local. You configure Ansible to interact with Kubernetes, and troubleshoot and fix the most popular errors. There is a lot of Ansible playbook code to create and report namespaces, deployments, pods, secrets, and persistent volumes. You'll apply multiple YAML files at once to Kubernetes. You also explore how to scale up and down your applications and run rolling updates.
- Chapter 5 explains how to build a web, RESTful Java microservice application using Spring technology and deploy it as a containerized image with the Tomcat web server. This chapter includes more use cases that explain how to automate the deployment of two examples of stateless (a PHP Guestbook application with Redis) and stateful (WordPress and MySQL with persistent volumes) applications in Kubernetes. You learn how to apply security to namespaces, pod resources (AppArmor), and pod syscalls (seccomp). You also learn how to save time using the Kustomize tool to generate Kubernetes YAML resources. Ansible Dynamic Inventory is handy for listing all the cluster resources that interact with Kubernetes API.
- Chapter 6 covers how to manage the control plane, which is a more useful tool for cluster administrator use cases, and discusses how to apply Ansible to the deployment of the DEV, SIT, UAT, and

PROD infrastructures. You learn how to deploy a monitoring tool (Prometheus) via the Helm package manager, fetch logs from resources for troubleshooting, and copy files and directories to and from a pod. The chapter also covers the processes of tainting, draining, cordoning, and uncordoning nodes. It also discusses rolling back deployments and DaemonSets and applying Security Authentication, Authorization, and Accounting (AAA).

- Chapter 7 is dedicated to the public could providers, including how you can take advantage of the hybrid/multi-cloud providers, what the shared responsibility model is, and some specific infrastructure implementations of Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure Cloud Services. It also covers how the Certified Kubernetes Conformance Program by CNCF is helping with portability of applications and services between different clusters.
- Chapter 8 is all about Ansible for enterprise, with the completed container-oriented Shiny Ansible Automation Platform providing the Web-UI RESTful API and Ansible Certified Collections. You learn about the future information technology trends of event-driven Ansible, Ansible Trusted Code, and Ansible Trusted Ansible Collections and Signed Ansible Projects. You also learn how artificial intelligence can propel your automation journey.

The sky is the limit with the OpenContainer standard, Kubernetes orchestration, and Ansible automation; you can deploy the cloud-native applications of tomorrow, taking advantage of unlimited infrastructure capacity and worldwide coverage.

Source Code

All source code used in this book can be downloaded from `github.com/Apress/Ansible-for-Kubernetes-by-Example-by-Luca-Berton`.

Disclaimer

Any opinions or personal views I express in this book are my own and not those of Red Hat Inc. or JPMorgan Chase & Co.

This work is not a product of JPMorgan Chase & Co. or its affiliates. Neither JPMorgan Chase & Co. nor any of its affiliates makes any explicit or implied representation or warranty regarding the contents of this paper and neither JPMorgan Chase & Co. nor any of its affiliates accepts any liability in connection with this work, including, without limitation, with respect to the completeness, accuracy, or reliability of the information contained herein, whether the contents of the paper violates or misappropriates the intellectual property of another, and the potential legal, compliance, tax, or accounting effects thereof.

CHAPTER 1

Modern IT Infrastructure and Hello App

Information technology infrastructure is evolving year after year, faster than ever. Following Moore's Law, servers have become more and more powerful and are able to execute more operations in a fraction of a second. Similarly, network infrastructure is evolving faster than ever.

The IT infrastructure is the backbone of a successful business. Companies efficiently manage their IT real estate with powerful tools that save time. The secret to success is a holistic cooperation between the IT Development and Operations teams. Automate, accelerate, and innovate are the mantras of every corporation in the post-pandemic world.

Kubernetes and Ansible are Infrastructure as Code (IaC) platforms that manage a fleet of services and make crucial decisions promptly.

Modern IT Infrastructure (DevOps and IaC)

All IT departments worldwide face challenges in efficiently delivering top-level customer IT services. In the same vein, it's important to obtain the most you can from your infrastructure (your return on investment). Just to be clear, the IT infrastructure is the whole data center: the servers, network devices, cabling, storage, and so on. Every good system administrator knows how important every piece in the infrastructure puzzle is. Traditionally, infrastructure teams in large organizations or system administrators in small ones carried the burden of manually managing the IT infrastructure, in order to deliver the best service for their internal and external stakeholders. A large organization typically operates multiple on-premise data centers and cloud providers.

Deploying an application in this traditional infrastructure required a combination of operating systems and underlying system libraries, as shown in Figure 1-1.

L. Berton, *Ansible for Kubernetes by Example*, https://doi.org/10.1007/978-1-4842-9285-3_1

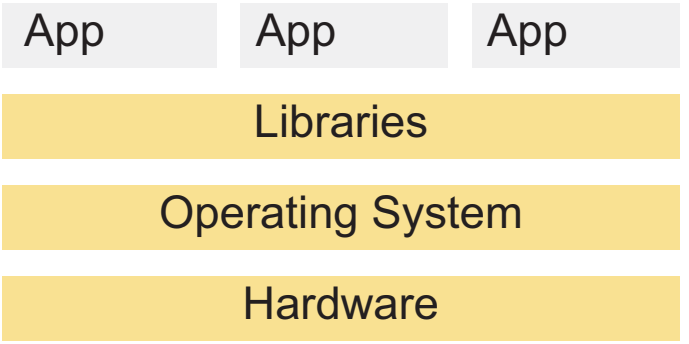

Figure 1-1. *Traditional deployment*

The main drawback of this architecture was that the applications were tightly connected to the underlying operating system and the system libraries. It was sometimes necessary to upgrade the underlying operating system just to update an application, or it was impossible to do so, because another application used a different version of the shared libraries.

The job of the infrastructure team included installing and updating the full lifecycle of the software, changing configurations, and managing services on servers, virtual machines, and nodes. The quality of the service provided could be measured using key indicators.

Some of you are familiar with key indicators, which reveal the uptime of software or hardware. Uptime measures the percentage of time the service is available to receive interactions by users. An uptime of 100 percent means the service is always available. Unfortunately, this was an unrealistic scenario because software and hardware failures negatively affects this in real life. Customers then experience a lower quality of service. You can apply this analysis to any web service, API service, website, web application, hardware, server, network device, or cloud provider. Some businesses also have penalties or insurance associated with poor uptime performance in their contracts.

For all those reasons, it's crucial to boost the performance of your infrastructure to accommodate business needs or to compete better in the marketplace.

Modern processor chips integrate virtualization technology, enabling resource splitting to accommodate increased demand. Static resource partition technology, such as VMware vSphere, allows many organizations to run multiple operating systems in one hardware setup. The ability to run various operating systems using virtualization, resource partition, Linux "cgroup" technologies, and Linux Containers (LXC) delivers virtual machines as base components of resource allocation. See Figure 1-2.

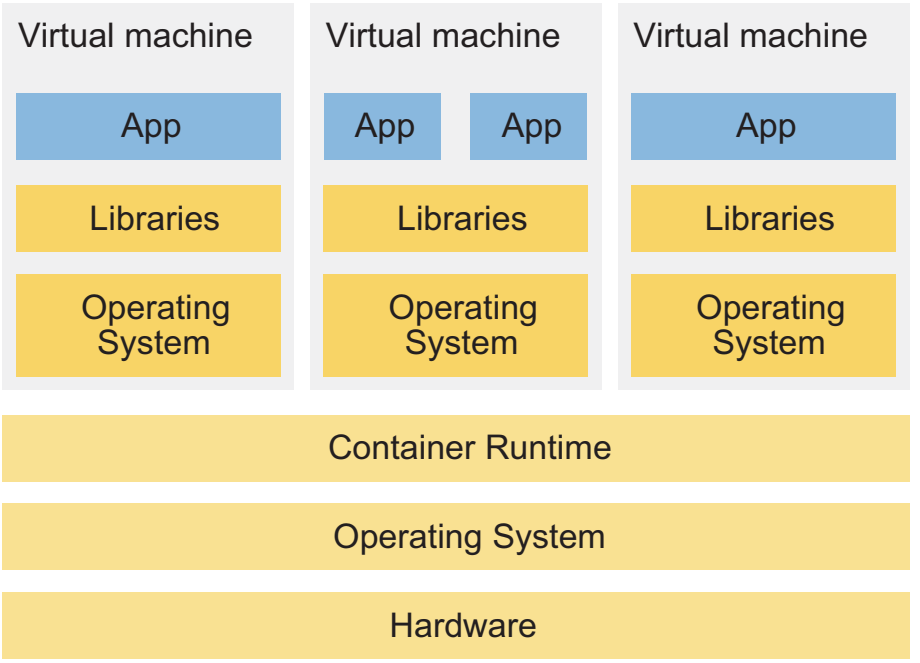

Figure 1-2. *Virtualized deployment*

Business flourished for well-known companies like VMware vSphere, Xen, and so on. Cloud providers such as Amazon Web Services (AWS), Azure Cloud, and Google Cloud Platform (GCP) also benefitted from the rise of virtualization. Their popular products are called Amazon EC2, Azure Virtual machines, and Google Compute Engine.

The most significant disadvantages of virtual machines are as follows:

- Static allocation of resources (CPU and RAM)
- More resources to manage by the IT infrastructure team
- Maintenance and overhead required of multiple operating systems (host and guest)
- Single hardware point of failure (mitigated by cluster premium features and the Veeam backup suite)

The Move to Containers

Due to those disadvantages, more organizations began to adopt a multi-cloud (multiple cloud vendors) or a hybrid-cloud (data center and one or more cloud vendors) strategy to offer better service to their customers. They could avoid the failure of one single vendor. Containers enable enterprises the flexibility to choose between on-premise, private, and public cloud providers or a mix of all these infrastructure technologies based on universal standards (SLA, quality of services, and cost). The portability of the application is ensured by the microservices software architectural design pattern, where applications are broken into their smallest components, independent from each other. See Figure 1-3.

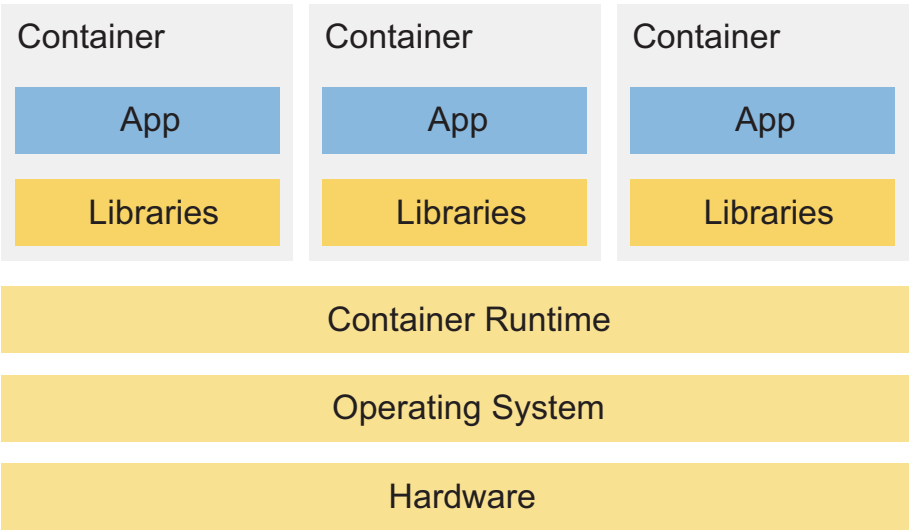

Figure 1-3. *Container deployment*

To deliver the best value in the least amount of time, IT engineers needed more powerful tools than bare metal or virtual machines. Deploying an application in a container is entirely independent of the underlying operating systems, as shown in Figure 1-3.

The Docker company was founded in 2011 by Kamel Founadi, Solomon Hykes, and Sebastien Pah, after the Y Combinator in the summer of 2010. Docker debuted to the public in Santa Clara at PyCon in 2013. Docker specifies how a container image is built and how the image is loaded and runs. Docker created a set of platform-as-a-service (PaaS) products that use OS-level virtualization to deliver containers, for example, the command-line Docker utility.

Podman is an open-source alternative to Docker (its full name is the POD MANager). Its main advantage is that you don't need a service to be running like Docker on your machine; Podman is "daemonless." The command syntax is very similar to Docker's. If you are concerned about security in your applications or planning to use Kubernetes, Podman is a better alternative. This book uses Docker because it's more well-known and widespread in the IT industry. But for most of the commands with the Docker prefix, you can substitute `docker` with `podman` and obtain the same result.

The template code of the container is called a *Dockerfile*, which allows you to build a binary version of the container, called an *image*. A more technology-agnostic alternative to Dockerfile is a *Containerfile,* by the Open Container Initiative (OCI), part of The Linux Foundation. Both are configuration files that automate the steps of creating a container image. The OCI promotes open industry standards around container formats and runtimes. The OCI releases three specifications: the Runtime Specification (`runtime-spec`), the Image Specification (`image-spec`), and the Distribution Specification (`distribution-spec`). Dockerfiles and Containerfiles use a combination of system tools, system libraries, code, runtimes, and settings to specify how an application runs. A single image can be run as many times as you want in a local Docker or Kubernetes cluster.

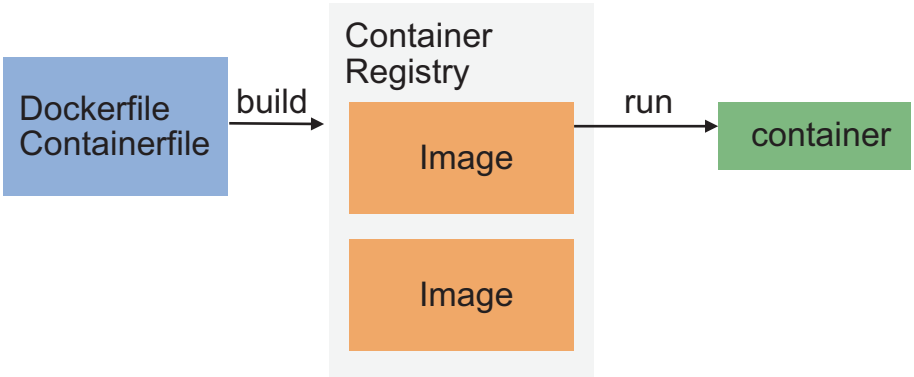

Figure 1-4. *Container lifecycle*

Once built, the container image is stored in a Container Registry (see Figure 1-4). The first Container Registry is Docker Hub, but many Container Registries are available in the marketplace nowadays. The first alternative that comes to my mind is Quay from Red Hat. Container Images are stored inside with metadata. That makes it easy for Kubernetes clusters to search within them and find them.

You can maintain multiple versions of your container in a Container Registry. You can also organize images in tags and branches that match the DEV (Development), SIT (System Integration Test), UAT (User Acceptance Test), and PROD (Production) DevOps principles.

Tags simply add a marker to a specific version of the image and make it easy for the users to find it.

If you need more than one container, you must move to a Docker Compose tool, which defines and runs multi-container Docker applications.

Modern business applications require one or more of the following features:

- High availability
- Multi-cloud compatibility
- Multi-tier storage
- Elastic/auto-scaling
- Self-healing
- Security by design (DevSecOps)

Containers can be useful for the rescue administrator workflow. In general, it's easier to manage storage, network, and configurations; containers help make developers more flexible and more productive.

First introduced by Docker, containers are a standard way to deliver applications and required libraries without the underlying operating system. This way, you can quickly relocate to another machine or promote the development to production.

These are the benefits of using containers:

- They are the standard way to package an application and library into one single object (container).
- They are much faster to spin up rather than using a virtual machine.
- They are easy to scale.
- They are portable because you can host anywhere.
- They enable microservices.

Running a container in a virtual machine is possible, but requires extra work to maintain the operating system of the running host and the guest container manually in a virtual machine. It is possible, but a sub-optimal solution.

Orchestration tools such as Kubernetes allow you to easily manage containers at scale.

Global trends of the DevOps and DevSecOps developer-centric methodologies deliver application-as-service, microservices, and serverless applications. They are ramping up faster and faster. In this scenario, containers created by the Continuous Integration and Continuous Delivery (CI/CD) toolset are tested for quality assurance and promoted to production faster than ever. Apache Mesos and Docker Swarm technologies are also often involved.

Ansible by Red Hat

Ansible is the leading open-source infrastructure automation technology. Infrastructure teams and system administrators worldwide realized that they needed a better way to scale their systems' management scripts to keep up with business demands. The hosted web applications increase the complexity, email flow, and new releases of the operating system. Manual work becomes unsustainable when the number of target devices grows. API-driven server management and configuration management tools like Ansible helped make things manageable.

The main Ansible use case is as follows:

- Provision
- Config management
- Application deployment

Red Hat Inc. (part of IBM since 2019) is leading the development of the Ansible project worldwide, guaranteeing the open-source quality of code.

The Ansible platform is available for a huge variety of modern operating systems and requires only an OpenSSH connection and a Python interpreter on the Linux target node. It supports a wide range of operating systems via different connection technologies. For example, you can use WinRM for Windows target nodes and Network API for network devices.

The learning curve is very rapid. You see this for yourself in Chapter 2, where you learn all about the Ansible programming language.

Red Hat also organizes the yearly AnsibleFest worldwide, usually at the end of October, to educate people about Ansible and share new features and success stories.

The Cloud Native Computing Foundation

In the early stage of application containerization, runtime systems such as Docker and Podman were extremely useful for manually deploying and testing applications.

However, when the workload became higher and managing failover was crucial, the world needed a more automated way of doing this.

Kubernetes (also called K8s) is the best way to orchestrate and manage a container workload. Kubernetes take care of deploying, scaling, and managing container applications. It is open-source and completely automatic.

Kubernetes has many releases per year; at the moment, there are three releases a year.

While Docker is focused on how a single container image is created, loaded, and run, Kubernetes is a way to schedule and manage a large number of containers.

You can deploy your container in Kubernetes to produce applications at a scale. You can seamlessly move containers and applications at scale. Similarly to Docker and Podman, you can run your container in Kubernetes. Kubernetes run anywhere—on-premise, in a private data center, in a public cloud, and even in embedded hardware such as the latest tiny Raspberry Pi devices. Kubernetes supports multiple processor architectures and modern operating systems.

Updating, upgrading, and managing in Kubernetes is relatively easy and guarantees security for your organization.

The Cloud Native Computing Foundation (CNCF) leads the development of the Kubernetes project, offering a vendor-neutral vision. CNCF is part of the Linux Foundation. CNCF also organizes community events, including KubeCon and CloudNativeCon, where you can meet experts and share stories.

The most inspiring customer stories are from innovation leaders such as BlackRock, Netflix, Zalando, Uber, *The New York Times,* ING, Squarespace, BlaBlaCar, Huawei, Amadeus, PNC Bank, Seagate, Verizon Media, and much more.

In Kubernetes, a *cluster* is made up of at least one Kubernetes master; many Kubernetes worker machines are called *nodes*. A cluster is fundamental for Kubernetes: all containerized workloads and services run within a cluster.

Kubernetes Support

Kubernetes relies on a vibrant community, external experts, and consultant services. Kubernetes is officially a self-support model, but more private organizations offer support and customization services.

Canonical, the organization leading the Ubuntu project, offers support for Kubeadm, MicroK8s, and Charmed Kubernetes on VMWare, OpenStack, bare metal, AWS, Azure, Google, Oracle Cloud, IBM Cloud, and Rackspace.

Kubernetes Distributions: OpenShift, Rancher, EKS, AKS, and GCP

There are currently several Kubernetes distribution options. They all integrate the core Kubernetes features, but they differ by the included modules and managing software. Kubernetes is the main framework for developing commercial products and cloud services.

Note This book focuses on the generally available Kubernetes, while most of the code also works in some cloud-native Kubernetes services.

OpenShift by Red Hat

OpenShift Container Platform (OCP) is a key product of the Red Hat multi-cloud strategy. It is getting traction in the market because it's easy to set up and maintain. Red Hat packages the core Kubernetes experience with a selection of valuable utilities to simplify the new user onboarding process. OpenShift Container Platform requires having a OpenShift subscription to use it, which includes Red Hat support. For this reason, it supports only Red Hat operating systems underneath, which is Red Hat Enterprise Linux (RHEL) and Red Hat CoreOS (discontinued).

Red Hat has a strong cloud-provider partnership for fully managed OpenShift services with products like Red Hat OpenShift Service on AWS (ROSA), Azure Red Hat OpenShift, and Red Hat OpenShift on Google Cloud Marketplace.

The main differences between OpenShift and Kubernetes are that OpenShift:

- Requires a paid subscription
- Has limited operating system support (Red Hat Enterprise Linux (RHEL) and Red Hat CoreOS
- Uses the `oc` command-line tool instead of `kubectl`.
- Has stricter security policies than default Kubernetes
- Forbids running a container as `root`
- Uses Role Based Access Control (RBAC) security
- Routes object instead of Ingress of Kubernetes based on HAproxy
- Uses ImageStreams for managing container images
- Uses OpenShift projects vs Kubernetes namespaces

OpenShift uses Helm as a software package manager, which simplifies the packaging and deployment of applications and services to OpenShift Container Platform clusters. OpenShift mainly uses Kubernetes operators; however, Helm is very popular in many Kubernetes applications. For Kubernetes-native applications, consider using Kubernetes Operator instead, because Operator continuously checks the application's status and determines if the application is running according to the configuration defined by the software developer.

Red Hat OpenShift Local (formerly Red Hat CodeReady Containers) is the fastest way to spin up an OpenShift cluster on your desktop/laptop for development purposes. This happens in minutes. It supports Linux (x86), Windows (x86), and macOS, even Intel and Apple Silicon (M1 and M2 processors). It is a simplified version relying on a virtual machine (~20GB) to set up, run, test, and emulate the cloud development environment locally. It uses the `crc` command-line utility to spin up a complete lab with an API and web interface. The requirements of the latest 2.10 release are four physical CPU cores, 9 GB of free memory, and 35 GB of storage space.

Red Hat also releases an open-source version of OpenShift called OKD. It's free to use and includes most of the features of its commercial product without support.

Kubernetes in the Public Cloud

Many cloud provider vendors build their Container Orchestration and Management products. The most popular cloud-native services are as follows:

- Amazon Elastic Container Service (Amazon ECS) by Amazon Web Services (AWS)
- Google Container Engine (GKE) by Google Cloud Platform (GCP)
- Azure Container Service by Microsoft Azure
- IBM Bluemix Cloud Kubernetes Container Service by IBM Cloud
- Oracle Container Cloud Service (OCCS) by Oracle Cloud
- Alibaba Cloud Container Registry

More services are available in the IT industry, but I consider these the major actors in the marketplace nowadays. See Chapter 7 for more details.

Amazon EKS

Amazon Elastic Kubernetes Service (Amazon EKS) is a service of Amazon Web Services (AWS). It adds the `eksctl` command-line tool for working with EKS clusters and uses the `kubectl` command underneath.

Amazon also provides Amazon Elastic Container Registry (Amazon ECR), an AWS-managed container image registry service that's cross-region and completely integrated within the AWS ecosystem. Specified users or Amazon EC2 instances defined in the IAM policies can access the ECR container repositories and images. Amazon ECR manages private image repositories consisting of Docker and Open Container Initiative (OCI) images and artifacts. Amazon Linux container images are already available in ECR, built with the same software components included in the Amazon Linux AMI for applications in Amazon EC2. See Chapter 7 for more details.

Containers and Pods

A *container* is a single unit for delivering software. But the real world is more complex; sometimes you need more than one container to provide the complete solution.