Hands-On Kubernetes, Service Mesh and Zero-Trust

Build and manage secure applications using Kubernetes and Istio

Swapnil Dubey

Mandar J. Kulkarni

www.bpbonline.com

First published: 2023

Published by BPB Online
WeWork
119 Marylebone Road
London NW1 5PU

UK | UAE | INDIA | SINGAPORE

ISBN 978-93-55518-675

www.bpbonline.com

Dedicated to

To my 'partners in Crime'
*(since childhood) : **Sneha, Shivam & Shubhanshu***

– Swapnil Dubey

****

My beloved wife:

Tejashri

&

*My Daughters **Rucha** and **Shreya***

– Mandar J. Kulkarni

About the Authors

- **Swapnil Dubey** has been working as an Architect at SLB since 2019, with a IT total experience of more that 14 years with enterpireses like Snapdeal, Pubmatic and Schlumberger. His current role at SLB involves designing and guiding technical teams implement data intensive workloads using Microservices and distributing computing architectural patterns hosted on public cloud (GCP & Azure) and On premise.

  In the past, he has served as Trainers for BigData technologies like Hadoop and Spark (Certified Trainer with Cloudera), and facilitated approximately 20 batches of people to kickstart their journey of Distributed computing. Moreover, he has spoken in multiple national & international conferences, where the key topic to talk was about containers and their management using Kubernetes.

  He completed his Masters from BITS Pilani in Data Analytics, and also holds Professional Architect Certifications in GCP and Microsoft Azure. This is his second book. Before this one, he has authored a book Scaling Google Cloud Platform with BPB Publications.

- **Mandar J. Kulkarni** has been working in software development and design for more than 16 years, and has played multiple roles such as Software Engineer, Senior Software Engineer, Technical Leader, Project manager and Software Architect. Currently, he is an architect in SLB building data products on top of Open Subsurface Data Universe (OSDU) Data Platform. He has also contributed to OSDU Data Platform with multiple architectural modifications and improvements.

  He has acquired Professional Cloud Architect certification from Google Cloud and also holds a Masters degree from BITS Pilani in Software Engineering. He has been a technical blogger for a while and this is first foray into writing a complete book.

About the Reviewer

Mahesh Chandrashekhar Erande has played the software architect role in the healthcare, telecom, and energy domains. For the past 19 years, he did end-to-end solution designing, programming and operationally supporting scalable enterprise apps. He is currently constructing the poly-cloud products for the SLB.

Acknowledgements

- Any accomplishment requires the effort of many people, and this work is no different. First and foremost, I would like to thank my family, (especially my father figure, mentor and guardian , Mr. N.R. Tiwari and My Mother – Sushma & Wife - Vartika) for continuously encouraging and supporting me in writing the book. I could have never completed this book without their support. Big thanks to the Energy which keeps pushing me everyday for my side hustles (apart from work).

  I gratefully acknowledge Mr. Mahesh Erande for his kind technical scrutiny of this book. My sincere thanks to the co author of the book, Mr. Mandar J. Kulkarni, whose constant enthusiasm and quality inspired me to bring out my best.

  My gratitude also goes to the team at BPB Publication for being supportive and patient during the editorial review of the book. A big thank you to SLB team for allowing me do this work.

  - Swapnil Dubey

- This book would not have been possible without continuous support from my family and friends. I thank them for their unconditional support and encouragement throughout this book's writing, especially my wife Tejashri and my brother Kedar.

  I am also grateful to the BPB Publications team for giving me the opportunity to author the book, and also for their support, guidance and expertise in making this book a reality. The participation and collaboration of reviewers, technical experts, and editors from team BPB has been very valuable for me as well as the book.

  Collaborating with author Mr. Swapnil Dubey has been an invaluable experience, and the learnings I gained, will guide me forever. I also want to thank Mr. Mahesh Erande for his technical reviews and feedback on the book content.

I would also like to acknowledge SLB for giving me the opportunities to work on the interesting technologies during my career and also for allowing me to write the book.

Finally, I would like to thank all the readers who keep taking interest in reading technical books. The appreciation and feedback from the readers is the biggest motivation for authors to create better content.

- Mandar J. Kulkarni

Preface

The objective of this book is to streamline the creating and operating workloads on Kubernetes. This book will guide and train software teams to run Kubernetes clusters directly (with or without EKS/GKS), use API gateways in production, and utilise Istio Service mesh, thereby having smooth, agile, and error-free delivery of business applications.

The reader masters the use of service mesh and Kubernetes, by delving into complexities and getting used to the best practices of these tools/approaches. While one runs hundreds of microservices and Kubernetes clusters, security is highly prone to be breached and that is where zero trust architecture would be kept in mind throughout the software development cycle.

The book also makes use of some of the great observability tools to provide a robust, yet clean set of monitoring metrics such as Latency, traffic, errors, and saturation to get a single performance dashboard for all microservices. After reading this book, challenges around application deployment in production, application reliability, application security and observability will be better understood, managed, and handled by the audience.

Chapter 1: Docker and Kubernetes 101 - This chapter will introduce the audience to the basics of Dockers and Kubernetes. In the docker section, the audience will get concepts to write and push images to container registries. We will give a walk through of an already developed application and package it in a docker container. There will be a discussion around practices which induce security vulnerabilities and their resolution. In the later part of the chapter, the audience will get introduced to Kubernetes, such as the why, what, and how of Kubernetes, followed by an in-depth understanding of architecture. There will be discussion around basic principles of Immutability, declarative and Self-healing way of assigning infrastructure in Kubernetes cluster.

Chapter 2: PODs – discusses the foundational block of Kubernetes called Pod. The chapter discusses the lifecycle of the pods along with health checks. The chapter also explains the resources requirements for Pod such as CPU, Memory as well as storage required for persisting data, along with security aspects like pod security standards and admissions.

Chapter 3: HTTP Load Balancing with Ingress - This chapter will discuss concepts of bringing the data in and out of an application deployed in Kubernetes. Ingress is a Kubernetes-native way to implement the "virtual hosting" pattern. This chapter will talk about exposing services deployed in Kubernetes to the outside world. AI gateways will also be discussed in this chapter taking example of open source API gateways like Gloo,Tyk and Kong. Apart from discussing the details around networking, readers will get the feel of security issues and loopholes which should be taken care of while configuring networking.

Chapter 4: Kubernetes Workload Resources – takes readers towards more practical examples of using Kubernetes in enterprise applications, by showing hands-on examples of creating workload resources such as deployments, replicasets, jobs and daemon sets. The chapter discusses the life cycle of each of these workload resources and explains which workload resource should be used for which use case while building scalable applications.

Chapter 5: ConfigMap, Secrets, and Labels - In this chapter, the concept of labels and secrets will be discussed. Labels can be used to select objects and to find collections of objects that satisfy certain conditions. In contrast, annotations are not used to identify and select objects. This chapter will help the audience to in-depth understanding of Annotations & Labels and strategies around how to use them effectively in real environments. This chapter will also help you understand the concepts of config map and a Secret better.

Chapter 6: Configuring Storage with Kubernetes – focuses on storage patterns with Kubernetes. The chapter discusses Volumes, Persistent volumes and stateful sets in details followed by a practical example of MongoDB installation. Furthermore, the chapter discusses disaster recovery of content stored using configured storage and the extesibility of Kubernetes architecture using container storage interface.

Chapter 7: Introduction to Service Discovery - Service discovery tools help solve the problem of finding which processes are listening at which addresses for which services. This chapter audience will get insight about various ways of discovering service in Kubernetes cluster. This chapter will act as a building block for section 3, where conceptual discussion will happen around how to achieve service discovery using Istio. The audience will also get insights into the various patterns of discovery and registration and the same will be showcased as hands-on exercises in the chapter.

Chapter 8: Zero Trust Using Kubernetes - This chapter will introduce the audience to the aspects of modelling and application with Zero trust principles in place. Lot of security aspects are already discussed in the previous chapters. For example, in Chapter 3, HTTP Load Balancing With Ingress, we will be talking about POD security. Similarly in Chapter 4, Kubernetes Worklad Resources, we plan to talk about security aspects when it comes to creation of networks. This chapter will give the audience a hands-on insight of how to achieve the aspects of this zero-trust security model using the individual building blocks discussed in the previous chapters.

Chapter 9: Monitoring, Logging and Observability - This chapter will talk about aspects of logging and monitoring of applications deployed in the Kubernetes cluster. This chapter will further discuss ways to implement basic SRE concepts and how the observability aspects are supported. Hands on exercises will demonstrate each of the concepts of logging, monitoring and SRE by enhancing the micro service application written and developed in earlier chapters.

Chapter 10: Effective Scaling - One of the key advantages of using Microservice deployed on Kubernetes is the power scaling mechanism. This chapter will help the audience understand the aspects of scaling in Kubernetes which includes horizontal & vertical pod scaling. Not only can we configure auto scaling on out of the box metrics, but also based on custom metric and combination of metrics. All the hands-on aspects will involve the three micro services which we created in earlier chapters. One Micro service will be planned to scale horizontally and vertically. Others will scale based on custom metrics, and third will showcase scaling based on a combination of two metrics.

Chapter 11: Introduction to Service Mesh and Istio – starts with the basics about microservices and then talks in details about the what, why and how of the service mesh concepts. The chapter discusses pros and cons of the service mesh as a concept and uses Isio as an example. The chapter then discusses Istio architecture, installation techniques and the customizations of Istio steup.

Chapter 12: Traffic Management Using Istio – is all about how to take the traffic management logic out of service code into the declarative yamls. The chapter discusses controlling ingress traffic, egress traffic and gateways. The chapter introduces Kubernetes's custom resources like VirtualService, DestinationRule, ServiceEntry and how to make use of them for achieving traffic management strategies like canary deployment, blue-green deployment. The chapter also explains with examples how to implement design patterns like circuit breaking, timeouts, retries and fault injection using service mesh like Istio. This chapter introduces and uses a sample application to explain the traffic management patterns.

Chapter 13: Observability Using Istio – talks about how different open source observability tools like Kiali, Grafana, Prometheus, Jaeger can be used alongside Istio to improve the observability. The sample application introduced in earlier chapters is used here again to show how to manage traffic patterns between different microservices, how to observe the scalability, how to monitor and search the logs, and how and where to view and search different metrics. The chapter also explains with examples how to use distributed tracing to debug latency issues in the application.

Chapter 14: Securing Your Services Using Istio – revolves around identity management, authorization and authentication using the built-in support that Istio provides. The chapter briefly introduces what is secure communication and then explains how Istio helps with Certificate management to make the intra-cluster communication secure by default. The chapter builds on top of the existing sample application used in previous chapters to explain concepts like permissive mode of Istio, Secure naming, Peer authentication, Service authorization, End-user authorization and so on. The chapter concludes by bringing it all together by explaining security architecture of Istio.

Code Bundle and Coloured Images

Please follow the link to download the
Code Bundle and the ***Coloured Images*** of the book:

https://rebrand.ly/l14igmh

The code bundle for the book is also hosted on GitHub at **https://github.com/bpbpublications/Hands-On-Kubernetes-Service-Mesh-and-Zero-Trust**. In case there's an update to the code, it will be updated on the existing GitHub repository.

We have code bundles from our rich catalogue of books and videos available at **https://github.com/bpbpublications**. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Piracy

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at **business@bpbonline.com** with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit **www.bpbonline.com**. We have worked with thousands of developers and tech professionals, just like you, to help them share their insights with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit **www.bpbonline.com**.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

Table of Contents

1. Docker and Kubernetes 101 .... **1**

Introduction 1

Structure 2

Objectives 2

Introduction to Docker 2

Introduction to Kubernetes 8

Kubernetes architecture *10*

Kubernetes Master *11*

Kubernetes Worker *14*

Principles of immutability, declarative and self-healing *16*

Principle of immutability *16*

Declarative configurations *16*

Self-healing systems *17*

Installing Kubernetes 17

Installing Kubernetes locally using Minikube *18*

Installing Kubernetes in Docker *19*

Kubernetes client 19

Checking the version *20*

Checking the status of Kubernetes Master Daemons *20*

Listing all worker nodes and describing the worker node *21*

Strategies to validate cluster quality 23

Cost-efficiency as measure of quality *23*

Right nodes *24*

Request and restrict specifications for pod CPU and memory resources *24*

Persistent volumes *24*

Data transfer costs and network costs *24*

Security as a measure of quality *25*

Conclusion 25
Points to remember 25
Multiple choice questions 26
Answers 26

2. PODs **27**
Introduction 27
Structure 28
Objectives 28
Concept of Pods 29
CRUD operations on Pods 30
Creating and running Pods 30
Listing Pods 31
Deleting Pods 33
Accessing PODs 34
Accessing via port forwarding 34
Running commands inside PODs using exec 35
Accessing logs 36
Managing resources 36
Resource requests: Minimum and maximum limits to PODs 36
Data persistence 38
Internal: Using data volumes with PODs 39
External: Data on remote disks 41
Health checks 42
Startup probe 42
Liveness probe 43
Readiness probe 43
POD security 44
Pod Security Standards 45
Pod Security Admissions 46

Conclusion 47
Points to remember 47
Questions 47
Answers *48*

3. HTTP Load Balancing with Ingress 49
Introduction 49
Structure 49
Objectives 50
Networking 101 50
Configuring Kubeproxy *53*
Configuring container network interfaces *54*
Ingress specifications and Ingress controller 55
Effective Ingress usage 62
Utilizing hostnames *62*
Utilizing paths *63*
Advanced Ingress 64
Running and managing multiple Ingress controllers *64*
Ingress and namespaces *64*
Path rewriting *64*
Serving TLS *65*
Alternate implementations 66
API gateways 68
Need for API gateways *68*
Routing requests *69*
Cross-cutting concerns *69*
Translating different protocols *69*
Securing network 69
Securing via network policies *69*
Securing via third-party tool *70*

Best practices for securing a network 71
Conclusion 72
Points to remember 72
Multiple choice questions 73
Answers 73
Questions 73

4. Kubernetes Workload Resources 75
Introduction 75
Structure 76
Objectives 77
ReplicaSets 77
Designing ReplicaSets 77
Creating ReplicaSets 78
Inspecting ReplicaSets 79
Scaling ReplicaSets 79
Deleting ReplicaSets 81
Deployments 81
Creating deployments 82
Managing deployments 83
Updating deployments 83
Deployment strategies 86
Monitoring deployment status 86
Deleting deployments 87
DaemonSets 87
Creating DaemonSets 87
Restricting DaemonSets to specific nodes 89
Updating DaemonSets 90
Deleting DaemonSets 91
Kubernetes Jobs 92

Jobs ... 92
Job patterns ... 94
Pod and container failures ... 94
Cleaning up finished jobs automatically ... 94
CronJobs ... 95
Conclusion ... 96
Points to remember ... 97
Questions ... 98
Answers ... 98

5. ConfigMap, Secrets, and Labels ... 99
Introduction ... 99
Structure ... 100
Objectives ... 100
ConfigMap ... 100
Creating ConfigMap ... 102
Consuming ConfigMaps ... 104
Consume ConfigMap in the environment variables ... 105
Set command-line arguments with ConfigMap ... 106
Consuming ConfigMap via volume plugin ... 107
Secrets ... 109
Creating Secrets ... 109
Consuming Secrets ... 111
Consuming Secrets mounted as volume ... 111
Consuming Secrets as environment variables ... 112
Private docker registries ... 112
Managing ConfigMaps and Secrets ... 113
Listing ... 113
Creating ... 114
Updating ... 114

Applying and modifying labels.......... 115

Labels selectors 117

Equality-based selector *117*

Set-based selectors.......... *118*

Role of labels in Kubernetes architecture.......... *118*

Defining annotations.......... 119

Conclusion.......... 120

Points to remember 120

Questions 120

Answers *121*

6. Configuring Storage with Kubernetes 123

Introduction.......... 123

Structure.......... 124

Objectives.......... 124

Storage provisioning in Kubernetes.......... 124

Volumes.......... *124*

Persistent Volumes and Persistent Volume claims *125*

Storage class.......... *130*

Using StorageClass for dynamic provisioning.......... *132*

StatefulSets.......... 133

Properties of StatefulSets.......... *133*

Volume claim templates.......... *137*

Headless service *137*

Installing MongoDB on Kubernetes using StatefulSets 138

Disaster recovery 140

Container storage interface 141

Conclusion.......... 142

Points to remember 142

Questions 143

Answers *143*

7. Introduction to Service Discovery 145

Introduction... 145

Structure.. 145

Objectives... 146

What is service discovery? 146

Client-side discovery pattern *148*

Server-side discovery pattern *150*

Service registry... 151

Registration patterns 151

Self-registration pattern.. *152*

Third-party registration... *152*

Service discovery in Kubernetes.. 153

Service discovery using etcd *153*

Service discovery in Kubernetes via Kubeproxy and DNS *157*

Service objects.. *159*

DNS... *160*

Readiness checks.. *160*

Advance details... 161

Endpoints.. *161*

Manual service discovery *163*

Cluster IP environment variables... *164*

Kubeproxy and cluster IPs... *164*

Conclusion 165

Points to remember 166

Questions 166

Answers *167*

8. Zero Trust Using Kubernetes.. 169

Introduction... 169

Structure.. 170

Objectives........170
Kubernetes security challenges........171
Role-based access control (RBAC)........173
Identity........173
Role and role bindings........174
Managing RBAC........177
Aggregating cluster roles........178
User groups for bindings........179
Introduction to Zero Trust Architecture........180
Recommendations for Kubernetes Pod security........182
Recommendations for Kubernetes network security........185
Recommendations for authentication and authorization........186
Recommendations for auditing and threat detection........187
Recommendation for application security practices........187
Zero trust in Kubernetes........188
Identity-based service to service accesses and communication........188
Include secret and certificate management and hardened Kubernetes encryption........189
Enable observability with audits and logging........190
Conclusion........191
Points to remember........192
Questions........192
Answers........193

9. Monitoring, Logging and Observability........195
Introduction........195
Structure........196
Objectives........196
Kubernetes observability deep dive........197
Selecting metrics for SLIs........199
Setting SLO........200

Tracking error budgets 200
Creating alerts 201
Probes and uptime checks 202
Pillars of Kubernetes observability 204
Challenges in observability 205
Exploring metrics using Prometheus and Grafana 206
Installing Prometheus and Grafana 208
Pushing custom metrics to Prometheus 211
Creating dashboard on the metrics using Grafana 213
Logging and tracing 214
Logging using Fluentd 215
Tracing with Open Telemetry using Jaeger 217
Defining a typical SRE process 220
Responsibilities of SRE 221
Incident management 222
Playbook maintenance 223
Drills 223
Selecting monitoring, metrics and visualization tools 224
Conclusion 225
Points to remember 225
Questions 226
Answers 226

10. Effective Scaling 227
Introduction 227
Structure 228
Objectives 228
Needs of scaling microservices individually 228
Principles of scaling 229
Challenges of scaling 230

Introduction to auto scaling ... 231
Types of scaling in K8s ... 232
Horizontal pod scaling ... 233
Metric threshold definition ... 237
Limitations of HPA ... 239
Vertical pod scaling ... 239
Cluster autoscaling ... 242
Standard metric scaling ... 244
Custom Metric scaling ... 247
Best practices of scaling ... 249
Conclusion ... 250
Points to remember ... 250
Questions ... 251
Answers ... 251

11. Introduction to Service Mesh and Istio ... 253
Introduction ... 253
Structure ... 254
Objectives ... 254
Why do you need a Service Mesh? ... 254
Service discovery ... 256
Load balancing the traffic ... 256
Monitoring the traffic between services ... 256
Collecting metrics ... 256
Recovering from failure ... 256
What is a Service Mesh? ... 257
What is Istio? ... 260
Istio architecture ... 261
Data plane ... 262
Control plane ... 263

Installing Istio....263

Installation using istioctl....264

Cost of using a Service Mesh....267

Data plane performance and resource consumption....267

Control plane performance and resource consumption....267

Customizing the Istio setup....268

Conclusion....269

Points to remember....270

Questions....270

Answers....270

12. Traffic Management Using Istio....273

Introduction....273

Structure....274

Objectives....274

Traffic management via gateways....274

Virtual service and destination rule....276

Controlling Ingress and Egress traffic....279

Shifting traffic between versions....280

Injecting faults for testing....284

Timeouts and retries....286

Circuit breaking....288

Conclusion....290

Points to remember....291

Questions....291

Answers....291

13. Observability Using Istio....293

Introduction....293

Structure....293

Objectives....294

Understanding the telemetry flow 294
Sample application and proxy logs 295
Visualizing Service Mesh with Kiali 297
Querying Istio Metrics with Prometheus 303
Monitoring dashboards with Grafana 305
Distributed tracing 308
Conclusion 313
Points to remember 314
Questions 314
Answers *315*

14. Securing Your Services Using Istio 317
Introduction 317
Structure 318
Objectives 318
Identity Management with Istio 318
Identity verification in TLS *319*
Certificate generation process in Istio *319*
Authentication with Istio 321
Mutual TLS authentication *321*
Secure naming *322*
Peer authentication with a sample application *323*
Authorization with Istio 327
Service authorization *328*
End user authorization *332*
Security architecture of Istio 336
Conclusion 337
Points to remember 338
Questions 338
Answers *338*

Index 341-347

CHAPTER 1
Docker and Kubernetes 101

Introduction

Software architecture evolves with time to cater to the needs of the latest industry workloads. For example, a few years ago, when the data size was insignificant, we used to write data processing workloads using multithreading, but the processing spanned across multiple machines. After that came the wave of Big data, where distributed computing frameworks like Hadoop and Spark were used to process huge volumes of data. We are now witnessing a similar wave. Today's architects believe in breaking a use case into more minor services and orchestrating user journeys by orchestrating the calls to the microservices. Thanks to Google for donating Kubernetes to the open-source world, such an architecture is now reality. With many organizations adopting Kubernetes for their infrastructure management, it has become the platform for orchestrating and managing container-based distributed applications, both in the cloud and on-premises. No matter what role you play in the organization, be it a developer, architect, or decision maker, it is imperative to understand the challenges and features of Kubernetes to design effective workflows for the organization.

Just like Kubernetes, Docker is one of the most widely used container runtime environments. Docker has seen its growth over the last few years, and while everybody agreed to the need for a container, there have always been debates about how to manage the life cycle of a container. The way Kubernetes and docker complement each other's needs makes them prominent partners for solving container-based workloads. Docker, the default container runtime engine, makes it easy to package an executable and push it to a remote registry from where others can later pull it.

In this chapter, you will dive deep into the concepts of Docker and Kubernetes. In the later chapters, one component discussed at a high level will be picked and discussed in further detail.

Structure

In this chapter, we will discuss the following topics:

- Introduction to Docker
- Introduction to Kubernetes
    - Kubernetes architecture
    - Principles of immutability, declarative and self-healing
- Installing Kubernetes
    - Installing Kubernetes locally
    - Installing Kubernetes in Docker
- Kubernetes client
- Strategies to validate cluster quality
    - Cost efficient
    - Security

Objectives

After studying this chapter, you should understand the basic working of Docker and Kubernetes. This chapter will also discuss some generic best practices when deploying docker and Kubernetes, and it will help you understand what factors you should keep in mind while enhancing reliability, resiliency, and efficiency better suited to the type of use cases you intend to solve. You will understand the principles of immutability, declarative, and self-healing, based on which the framework of Kubernetes stands. This chapter will also help you learn how to evaluate the quality of your cluster as per the needs of use cases.

Introduction to Docker

Docker is the most widely used container runtime environment; it enables creating containers and running them on some infrastructure. Some infrastructure could be physical on-premise nodes or virtual machines on any cloud platform. Developing, shipping and running applications are key terms when discussing docker. You can develop applications with each application having its binaries and libraries, and package them by creating an image. These images could be instantiated by running them as containers. Each container with separate applications can run on the same physical or virtual machine without impacting the other.

Consider *Figure 1.1*, which demonstrates the preceding discussion in detail.

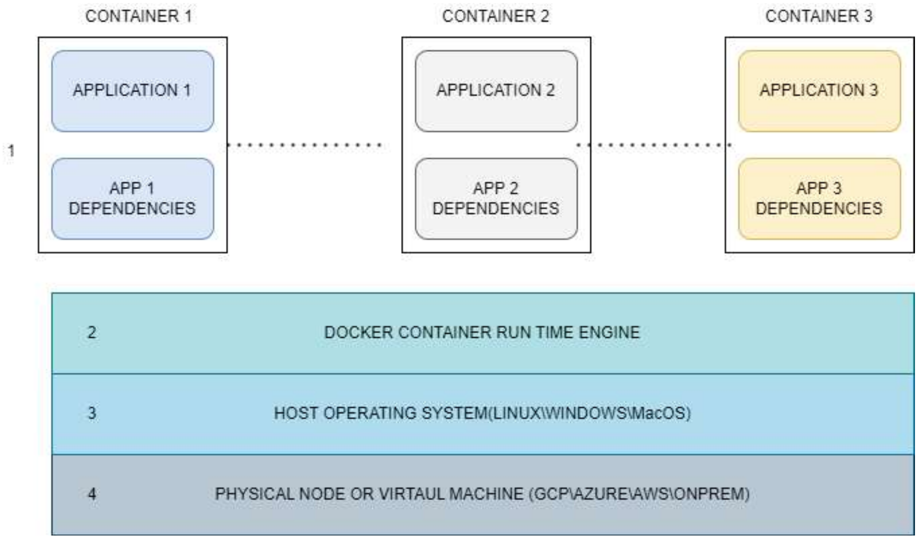

Figure 1.1: *Docker 101*

Refer to the numerical labelling in *Figure 1.1* with the following corresponding numerical explanations:

1. Multiple applications with completely different technology stacks could be developed, and their complete dependencies could be packaged as container images. These container images, when instantiated, become containers.
2. These container images need a container runtime environment. The container runtime environment provides all the features to launch , execute, and delete an image. Multiple runtime environments are available in the industry, such as runC, containerd, Docker, and Windows Containers.
3. These container runtime environments run on top of any operating system. For example, a docker runtime could be installed on Linux, Windows, or macOS, unless the container runtime is installed successfully and no other host operating system restrictions apply.
4. The mentioned setup can run on a physical or virtual machine, and on-premise machines on public cloud providers like GCP, AWS, or Azure. In fact,

the setup can run on a device in which an operating system and **Container Runtime Environment** (**CRE**) could be installed.

With this basic introduction to how containers, container run time, and physical infrastructure align, let's now look at Docker precisely and understand the game's rules with Docker as a container runtime environment. *Figure 1.2* demonstrates the docker process of building and using images:

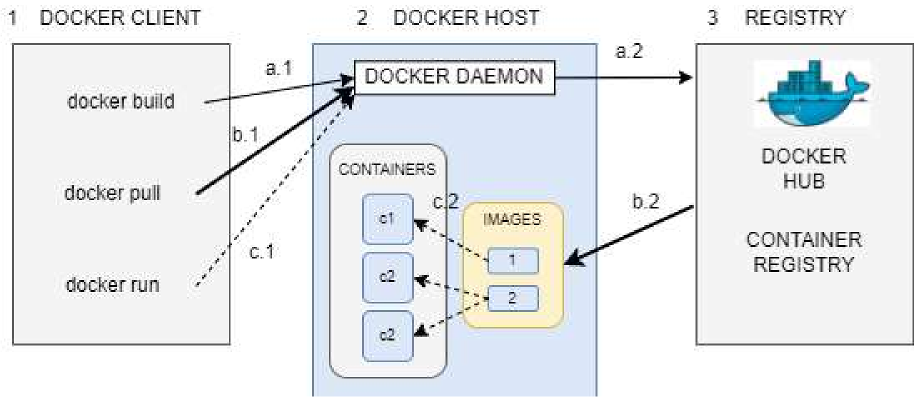

***Figure 1.2**: Docker Process*

Refer to the numerical labelling in *Figure 1.2* with the following corresponding numerical explanations:

1. Docker client is a CLI utility to execute docker commands. In this section, there are three main commands that everybody should know about.
2. The docker daemon interprets the CLI commands, and the action is performed.
3. A registry is a repo where you build and upload the image. A few standard registries are docker hub, quay.io, and registries with cloud providers, such as Google Container registry in the Google cloud platform.

Let us talk about the three docker commands shown in *Figure 1.2*:

- **`Docker builds <docker-file-path>`**: You specify the contents of your repo in a plaintext file (which are written as per the construct suggested by Docker). This command creates a local image using the plaintext file(created above). Look at the arrows labeled with **`a`**.
  - **`a.1`**: Docker build command is interpreted by the docker daemon.
  - **`a.2`**: The image created by the docker build command can be pushed to container registries.

- **Docker pulls <container registry link for image>**: This command pulls the image from the registry to a local machine. Look at the thick solid lines and follow the flow labelled as **b**.
    - **b.1**: The docker daemon interprets the docker pull command, and a pull call is made to a remote repository.
    - **b.2**: Docker image is pulled to a local system.
- **Docker run <image name>**: Create a container using one of the docker images available locally in the systems.
    - **c.1**: Docker run command interpreted by docker daemon.
    - **c.2**: Containers are created using images. Image labeled as one is used in creating container **c1**, and image two is used in creating containers **c2** and **c3**.

Now is the time to investigate the complete preceding defined process. For this exercise, refer to the **docker-demo-python-app** folder in the code base attached to this chapter. It is a simple hello world Python application. If you look at the folder's contents, there are python-related files and a file named **Dockerfile**.

You will use docker hub, an openly available container registry, for this exercise. Follow the given steps:

1. **Log in to the docker hub**

   Type the following command and enter your username and password for the docker hub. To create this username and password, get yourself registered at **https://hub.docker.com/signup**.

   ```
   $ docker login
   ```

   Refer to *Figure 1.3*:

```
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a D
Username: swapnildubey1984
Password:
WARNING! Your password will be stored unencrypted in /home/sdubey7/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

Figure 1.3: *Docker hub Login*

2. **Build an application image**

   In this step, you will build the docker image in local. We will discuss how a docker file looks in the next step.

   ```
   $ docker build -t demo-python-app:1.0.1 .
   ```

Once the preceding command completes, run the following command if your docker image is present locally:

`$ docker images|grep 'demo-python'`

3. **Build multistage images**

It is time to investigate the docker file you used to create an image.

```
FROM python

# Creating Application Source Code Directory
RUN mkdir -p /usr/src/app

# Setting Home Directory for containers
WORKDIR /usr/src/app

# Installing Python dependencies
COPY requirements.txt /usr/src/app/
RUN pip install --no-cache-dir -r requirements.txt

# Copying src code to Container
COPY . /usr/src/app


# Application Environment variables
#ENV APP_ENV development
ENV PORT 8080


# Exposing Ports
EXPOSE $PORT


# Setting Persistent data
VOLUME ["/app-data"]


# Running Python Application
CMD gunicorn -b :$PORT -c gunicorn.conf.py main:app
```

In the preceding file, you can see the first line, **`FROM python`**, meaning that this image, when built, will first pull the Python image and then prepare a new image by adding the following details in the **`Dockerfile`**.

This is known as multistage pipelines, and there are obvious advantages. You can build an image once and then reuse and share the same image as sub images in across multiple images. For example, in your Enterprise, there could be one hardened image by security team for Python, and all teams could use the hardened Python image and use it to create application code specific image.. This makes the **`Dockerfile`** creation simple and more straightforward.

Also, note the constructs like **`RUN`**, **`COPY`**, **`EXPOSE`**, and so on. These are docker-specific constructs and have a special meaning in the docker container runtime environment.

4. **Store images in registries**

The image **`demo-python-app:1.0.1`**, which you built in step 2, is still available locally, meaning that no other machine can create a container using that image. For this, you have to share the image with the container registry. You will be using the docker hub for this. Once you have an account created and have logged in to docker hub, you can trigger the following two-step process to push the image:

i. **Step 1**: Tag the image

```
$ docker tag demo-python-app:1.0.1 <dockerhub-username>/
demo-python-app-dockerhub:1.0.
```

ii. **Step 2**: Push the image to docker hub

```
$ docker push <dockerhub-username>/demo-python-app-
dockerhub:1.0.
```

On docker hub web page, you can see if the image is pushed or not. Refer to *Figure 1.4*:

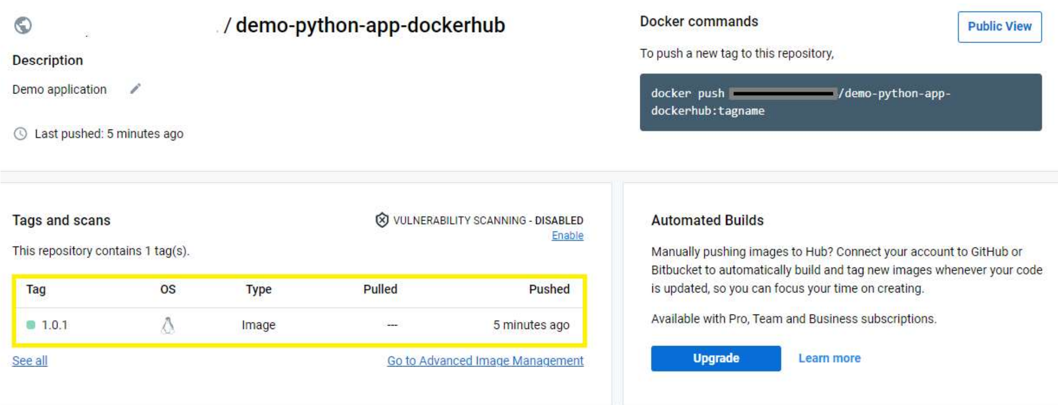

Figure 1.4: *Docker Hub Repo*

5. **Container runtime**

   The docker image pushed to the docker hub can now be pulled into any machine having docker installed. The pull of the image will make a copy from the remote repo to the local machine, and then the docker image can be executed.

```
$ docker pull <dockerhub-username>/demo-python-app-
dockerhub:1.0.1
```

```
$ docker images| grep "demo-python"
```

   The preceding docker image command will now show two results: the local one, that is, **`demo-python-app:1.0.1`**, and **`demo-python-app-dockerhub:1.0.1`**. Refer to *Figure 1.5*:

```
demo-python-app                              1.0.1    aafe172b02ee   24 minutes ago   951MB
[redacted]/demo-python-app-dockerhub         1.0.1    aafe172b02ee   24 minutes ago   951MB
```

Figure 1.5: *Local Docker Images*

   As the last step, you can create a docker container using the following command:

```
$ docker run -d -p 8080:8080 <dockerhub-username>/demo-python-app-
dockerhub:1.0.1
```

Open the web browser and feed the URL **`localhost:8080`**; a web page will open, and this will show that the container is created and exposed at port **`8080`** of the machine.

Introduction to Kubernetes

Kubernetes is an open-source container orchestrator for efficiently deploying and hosting containerized applications. Google initially developed it as an internal project to host scalable distributed applications reliably. In modern software systems, services are delivered over the network via APIs. The hosting and running of the APIs generally happen over multiple machines present geographically at the same or different locations. Also, since the data is growing every day, the scalability aspect of such services has started taking center stage, with no point in service-delivering responses breaching **Service Level Agreements** (**SLA**). Your application should use the optimal infrastructure to keep costs in check. Both the aspects of applications, that is, being scalable (up and down) and distributed, make sense only when the system is reliable. An algorithm running on such modern systems should produce the same results in multiple runs without any dependence on where and how the application is hosted.

Since Kubernetes was made open-source in 2014, it has become one of the most popular open-source projects in the world. Kubernetes APIs have become the de facto standard for building cloud-native applications. Kubernetes is a managed offering from almost all cloud providers: Google cloud platform, Amazon Web Services, and Microsoft Azure. Kubernetes, also known as K8S, automates containerized applications' deployment, scaling, and management. It provides planet-scale infra; if you keep supplying physical infrastructure, Kubernetes can scale up your application to significant levels. The larger the deployment, the greater the chance of parts of the infrastructure failing; Kubernetes has auto-healing properties, enabling automated recovery from failures.

Kubernetes also has some extremely mature features apart from the ones already mentioned. A few of the handy ones are as follows:

- **Capability to scale**: The application deployed in Kubernetes can scale horizontally (scaling up and down) and vertically (scaling in and out).
- **Security**: Kubernetes provides a platform for secured communications between multiple services. The extent depends on the type of application, for example, applying authentication and authorization on the services accepting internet data (external, front facing) to user authentication and consent to all services (internal and external)
- **Extensibility**: This refers to adding more features to the Kubernetes cluster without impacting the already present applications. For example, you can integrate plugins that will produce metrics about your application that are needed to perform SRE activities.
- **Support for batch executions**: We have only discussed services so far; however, Kubernetes provides support for executing batch jobs and also provides the ability to trigger cron jobs.
- **Rollbacks and roll-outs**: Kubernetes support features to roll back and roll out your application in stages, meaning that you can choose to deploy a new version of the service by just allowing it to serve 10% of users and then allow it for all.
- **Storage and config management**: Kubernetes provides the capability to use various storage solutions – SSD or HDD, Google Cloud Storage, AWS S3, or Azure Storage. In addition, Kubernetes has support for effectively managing general and secret configurations.

In the current book, you will see the preceding features being described and explained in depth, with special attention to security aspects and production readiness of the Kubernetes platform and applications.

Kubernetes architecture

Kubernetes is a complete and competent framework for modern workloads, but it is also very complex. When they read the documentation, many people get overwhelmed and get lost in the amount of information it provides. In this section, you will see the architecture of Kubernetes, and we will talk about the basics of the architecture, its components, and the roles each component plays in how Kubernetes does what it does.

Kubernetes follows a master-worker architecture. Consider *Figure 1.6*; you will see components - worker nodes and master nodes. As the name suggests, worker nodes are where actual work happens, and the master node is where we control and synchronize the working between worker nodes:

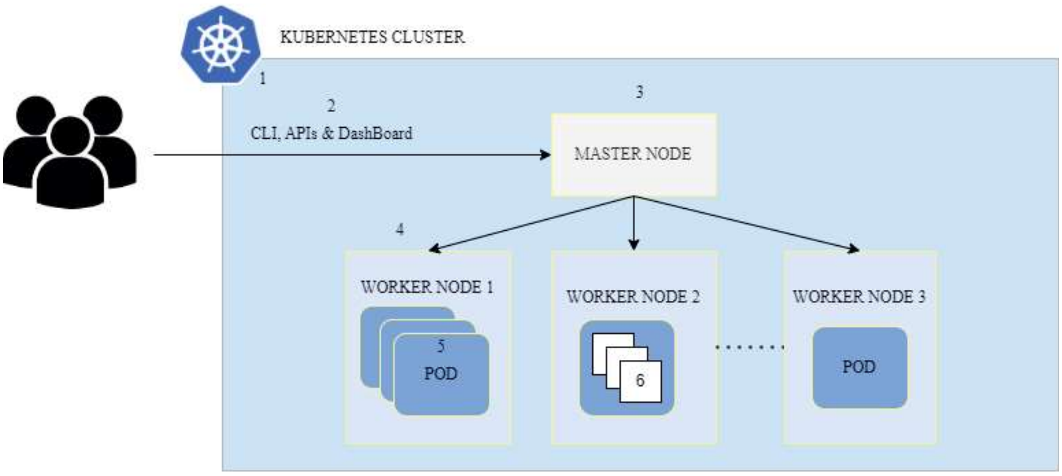

Figure 1.6: *Kubernetes 101*

Refer to the numerical labeling in *Figure 1.6* with the following corresponding numerical explanations:

1. Label 1 represents a complete Kubernetes cluster. A Kubernetes cluster is a collection of physical machines/nodes or virtual machines, with an official limit of a max of 5000 nodes. The control plane (Master) and workload execution plane (Worker) are deployed on these nodes. The expectation from the nodes comes from expectations from the Kubernetes components. For example, your master machine could only be a Linux box, while the worker nodes can be windows boxes too.
    a. Kubernetes is responsible for identifying and keeping track of which nodes are available in the cluster. Still, Kubernetes does not manage the node, which includes things like managing the file system,

updating the operating system security patches, and so on, inside the node. The management of the node becomes the responsibility of a separate components/team.

2. Any system/entity or person (developer or admin) can interact with the Kubernetes cluster via CLI, APIs, and Dashboard. All these interactions happen only via Master nodes.
3. The master node manages and controls the Kubernetes cluster and is the entry point for all admin tasks. Since the master node is responsible for maintaining the entire infrastructure of the Kubernetes cluster, when master nodes are offline or degraded, the cluster ceases to be a cluster. The nodes are just a bunch of ad hoc nodes for the period, and the cluster does not respond to the creation of new resources(pods), node failures, and so on. No new workloads can be launched on the cluster.
4. Worker nodes are the workhorses of the cluster, which perform the actual processing of data. They only take instructions from the Master and revert to the Master. If they do not receive any signal from the Master, they will keep waiting for the following instructions. For example, in the scenario of Master being down, the worker node will finish the work running on them and will keep waiting. If a worker node is down, it results in the low processing capability of the cluster.
5. Kubernetes Pods host workloads. A workload and all its supporting needs, like exposing ports, infrastructure and other networking requirements, and so on, are specified in a YAML file. This file is used to spin up a container or multiple containers in a Pod. Since you define one YAML per pod and each pod can have multiple containers, all containers share the resources inside a pod. For example, if your workload creates two containers inside the pod and your YAML file assigns them, both pods will share this one core of the CPU.
6. Containers represent the containerized version of your application code. Containers inside one pod are co-located and co-scheduled to run on the same Kubernetes work node. Kubernetes support multiple container runtime environments like containerd, CRI-O, docker, and several others. You will see docker being used throughout the book.

With the preceding behavioral concepts in mind, let us look at the components and internal working of the Master and worker nodes.

Kubernetes Master

Kubernetes Master, or the control plane of the Kubernetes cluster comes to life when a Kubernetes cluster is created and dies when a cluster is deleted. Generally, the